سلسلة بإشراف
د. عبد الحسن الحسيني

الان تيبسران

لغة باسكال

ترجمة د. محمد الحجار

المؤسسة الجامعية للدراسات والنشر والتوزيع

لغة باسكال

الطبعة الأولى

1408 هـ ـ 1988 م



بيروت ـ الحمراء ـ شارع اميل اده ـ بناية سلام
هاتف ٨٠٢٤٢٨ ـ ٨٠٢٤٠٧ ـ ٨٠٢٢٩٦
بيروت ـ المصيطبة ـ بناية طاهر هاتف : ٣٠١٠٣٠ ـ ٣١١٣١٠
ص . ب : ١١٣/٦٣١١ تلكس LE ٢٠٦٦٥ ـ ٢٠٦٨٠ لبنان

هذا الكتاب ترجمة :

Pascal iso / afnor

Programmation déductive
et description de la norme

Par
Alain TISSERANT

مقدمة

بين اللغات الموضوعة لحل أزمة مناهج السبعينات ، كان للغة الباسكال النجاح الأكبر . بعد تنظيمها ووضعها تقريباً في كل الحاسوبات ، من الميكرو الى الكبير ، يتّسع إستعمالها يوماً بعد يوم مع وجود برامج تمتد من 10 أسطر الى 000 800 سطر في السنترالات التلفونية .

إنّـها أكثر سهولة من الكوبول أو الفورتران ، لكن كذلك أسرع وفعالة أكثر من أجل إبراز مفهوم ، تنقيح وصيانة البرامج .

الباسكال هي لغة للمبتدئين ، لقد تم تصوّرها بالفعل ، لتعليم البرمجة كما لو كانت مادة تعليمية دقيقة قائمة على بعض المفاهيم الأساسية الظاهرة بوضوح في اللغة .
الباسكال هي لغة للمحترفين ، إنها فعالة جداً عند التنفيذ ، وتسمح بالكتابة الجيدة لبرامج تُقـرأ بسهولة ، فإذن قابلة للتعديل من شخص آخر وسهلة التنقيح .

لدى هذا العمل ، الطموح بأن يكون قد استجاب لطلبات الممتهنين وكذلك قد استُفيد منه كدعامة للمادة التعليمية . لكي يكون فعالاً إعتمدنا العرض التـدريجي لذا نراه ، يحتوي على تكريرات مفتعلة ، لكن يستخدم أسلوباً وتنويطات محددة . نجد فيه الكثير من الأمثلة ، لكن القليل من التفصيلات التكنولوجية .

تتطابق اللغة المشروحة مع النُظم ISO وAFNOR

يسمح الجزء الأول للمبتدىء ، بأن ينطلق في عالم البرمجة مستعملاً طريقة أثبتت وجودها . نشرح في الجزء الثاني الأدوات المُعالَجة ومن ثم في الجزء الثالث طريقة مُعالجتها ؛ إن العناصر المُدخلة في هذين الجزئين تتماشى مع كل ما هو ضـروري للبرمجـة في جميع الأوقات . يُتمِّم الجزء الرابع المفاهيم التي سبقته بهدف إستعمال لغة الباسكال بشكلها الكامل . أخيراً ، يوجز الجزء الخامس اللغة الى مُذكِّـرة مساعِدة . تقوم بعض الملحقات بإضافة بعض المتمِّمات ، وتسهِّـل البلوغ المباشر في الكتاب .

لقد عرَفت لغة الباسكال ، المحدّدة عام 1969 من قبل نيكلوس ويرث Niklaus wirth كما لو كانت تطبيقاً للبرمجة المركّبة، إنتشاراً سريعاً . اليوم يوجد نظام عالمي يحدِّد اللغة ؛ مع ذلك فإن معالجي الباسكال الموجودين لا يخضعوا جميعهم لهذا النظام ؛ فإذن أصبح من الواجب على القارىء ، إدخال تعديلات طفيفة على البرامج المعطيّـة كأمثلة ، قبل إستعمالها على حاسوبه المفضل .

الفصل الأول

كيفية البدء بكتابة البرنامج

0.1 ـ التحليل

يراد من هذا الفصل أن يكون مدخلاً للمبتدئين في البرمجة : قبل الدخول في تفاصيل كتابة البرامج في لغة الباسكال «Pascal» ، أو في أية لغة أخرى ، يجب الإلمام ببعض مبادىء التحليل .

تعني كلمة التحليل في عرضنا هذا فن العبور بمشكلة الى برنامج يحلُّها . إن ذلك يستوجب أنشطة عقلية ليست دائماً سهلة لكنها ضرورية ، والتي يمكن جعلُها سَلِسَةً أكثر بوضع أنفسنا ضمن إطار محدد : الطريقة الإستنتاجية . إنها طريقة شديدة السهولة بحيث تتيح تعليم سريع ، كما أنها فعّالة بحيث تسمح بإدخال المفاهيم العامة والمفردات والمصطلحات الأساسية للبرمجة .

سيتم تناول كتابة البرامج في لغة الباسكال بالمعنى الحرفي للكلمة ، في الفصل الثاني . هذا لا يمنع الحصول منذ الآن على برامج « تعمل » .

1.1 ـ الإنطلاق من النتيجة

لنفرض المسألة التالية : « أَجْرِ فوترة على ميزان مسجِّل » . بطرحها بهذه الطريقة ، فإن المسألة تجزّنا إلى عدة أسئلة : ما هي وحدات القياس المستعملة (كيلوغرام ، طن ؟) ، كيف يتم حساب الثمن المتوجب دفعه عند إتمام الوزنة (هل هناك من مبلغ إضافي ثابت ، ضريبة ، حسم ؟) ، كم يوجد من وزنات متتالية ، إلخ . . إن الإجابة على هذه الأسئلة تعني العبور من المشكلة (المسألة الأساسية) إلى المواصفة أي حل مُضْمَرات المسألة .

للإجابة على الأسئلة ، يجب العمل على استدعاء مجموعة من المعلومات الخاصة بالمسألة المطروحة ؛ يمكن أن ينتج عن ذلك أنه من الضروري مراجعة كتاب ما ، أو مراجعة متخصّص في ذاك المضمار (الرياضيات ، المحاسبة ، الميكانيك ، . . .) . إذا كانت المواصفة غير كاملة أو مغلوطة ، فإن البرنامج لن يؤدي بالتالي الى نتائج جيدة .

يمكن أن تكون المواصفة في مثلنا هذا هي التالية : « إحسب الثمن المتوجب دفعه (عدد صحيح من القروش) بعد إجراء وزنة ؛ الثمن هو حاصل ضرب الوزن (عدد صحيح من الغرامات) بالسعر الإفرادي (عدد صحيح من القروش للكيلوغرام الواحد) » .

إن المواصفة تحدد نصوص المسألة والحسابات الواجب إجراؤها لحلِّها ، لكنها غير معنية بتفاصيل الحسابات ، ولا بترتيب العمليات : إن هذا هو هدف كتابة التحليل الذي يُترجم فيما بعد وبطريقة سهلة جداً وأوتوماتية إلى برنامج .

مسألة ← مواصفة ← تحليل ← برنامج

التحليل

يتألف التحليل من تعريفات ، معادلات تُعيِّن قيمة إنطلاقاً من ثوابت وقِيَم أخرى (الثمن = الوزن * السعر الإفرادي / 1000) ، ومن معجم ، فهرس لأسماء (= معرَّفين) معطيَّة للقيم المتوجب حسابها :

التعريفات		المعجم
نتيجة = « اكتب » ثمن	3	
الثمن = الوزن * السعر الإفرادي / 1000	2	ـ الثمن (صحيح) : الثمن المتوجب دفعه من قبل الزبون ، بالقروش ـ الوزن (صحيح) : الوزن بالغرام
الوزن ، السعر الإفرادي وفقاً لمعطيّة	1	ـ السعر الإفرادي (صحيح) : السعر بالقروش للكيلوغرام

↑ ترتيب التعريفات

الطريقة الإستنتاجية (Méthode déductive)

لتكوين تحليل ، تقترح الطريقة الإستنتاجية الإنطلاق من تعريف النتيجة للصعود نحو المعطيات .

القاعدة 1 :	نُحدِّد « النتيجة » ؛ إن تعريفها هو العنصر الأول الذي يجب إدخاله في التحليل ؛
القاعدة 2 :	ومن ثم ، « في كل مرحلة » ، نستبدل أحد المعرِّفين الذين لم يحددوا بعد من المعجم بـ : ـ في قسم التعريفات ، تعريف المعرِّف ، وذلك ، إذا كان ضرورياً ، بإدخال معرِّفين جدد ـ في قسم المعجم ، المعرِّفين الجدد ، وذلك حتى يصبح كل معرِّفي المعجم قد تم تحديدهم .
القاعدة 3 :	في النهاية نحدد ترتيب التعريفات (من المعطيات إلى النتيجة) .

(نستنتج من كل تعريف مجموعة أسئلة التي بدورها تؤدي الى تعريفات جديدة) .
هكذا فإن الترتيب الأوّلي المتّبع في كتابة التعريفات لا يهمه أمر الترتيب النهائي المتّبع في عملية الحساب .

يمكن أن يكون التعريف خاصاً بقيمة داخلية ، مثلاً الثمن = الوزن * السعر الإفرادي ، أو خاصاً بقيمة خارجية تكون إما نتيجة (نتيجة = «اكتب» ثمن) ، إما معطيّة (الوزن ، السعر الإفرادي « وفقاً » لمعطيّة) .

فيما يخص المعجم ، فإنه يتم وصف كل معرِّف بـ :

- نوع القيمة الذي يمثل (صحيح ، حقيقي ، سمة ، . . .)
- وبملاحظة موجزة على أن يتم تحديد إستعمالها .

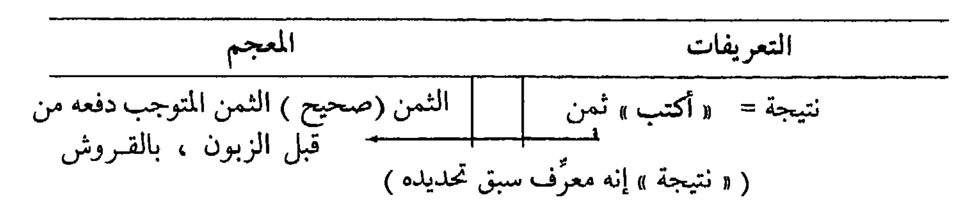
فيما يخص التعبير ، القسم الأيسر من التعريف ، نجد :

- مجموعة معرِّفين ، مثل سعر ، وزن
- مجموعة ثوابت ، إمّا صحيحة مثل 100 ، 7 −
  إمّا حقيقية مثل 3.14159 ، 0.05 −
- مجموعة مؤثرات ، مثل + ، − ، * (عملية الضرب) div (قسمة صحيحة : النتيجة تكون قيمة صحيحة) .

للمعرِّف تعريف وحيد ، لذا يتم في المعجم ، شطب المعرِّفين الذين سبق تعريفهم .

مثال

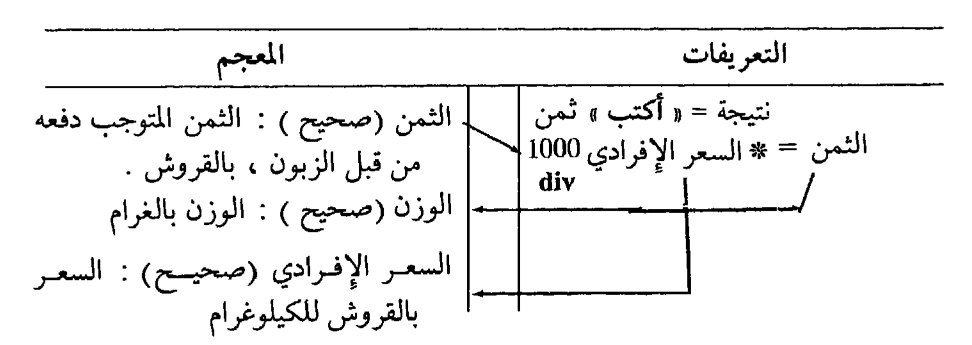
المرحلة 1 القاعدة 1 (تعريف « النتيجة »)



المعجم	التعريفات
الثمن (صحيح) الثمن المتوجب دفعه من قبل الزبون ، بالقـروش	نتيجة = « أكتب » ثمن

(« نتيجة » إنه معرٌف سبق تحديده)

المرحلة 2 : القاعدة 2 (تعريف الثمن)



المعجم	التعريفات
الثمن (صحيح) : الثمن المتوجب دفعه من قبل الزبون ، بالقروش .	نتيجة = « أكتب » ثمن
الوزن (صحيح) : الوزن بالغرام	الثمن = * السعر الإفرادي 1000 div
السعـر الإفـرادي (صحيـح) : السعـر بالقروش للكيلوغرام	

المرحلة 3 : القاعـدة 2 (تعريف الوزن والسعر الإفرادي)

المعجم	التعريفات
الثمن (صحيح) الثمن المتوجب دفعه من قبل الزبون ، بالقروش	نتيجة = « أكتب »: ثمن
الوزن (صحيح) : الوزن بالغرام	الثمن = الوزن * السعر الإفرادي div 1000
السعـر الإفـرادي (صحيـح) : السعـر بالقروش للكيلوغرام	الوزن ، السعر الإفرادي « وفقاً » لمعطيّـة

المرحلة 4 : القاعدة 3

كل معرّفي المعجم قد تم تعريفهم ، بذلك نعمل على ترتيب التعريفات :

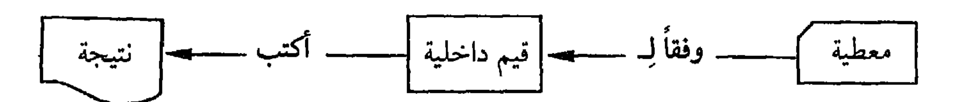
« قاعدة : لا يمكن إستعمال معرًف (في القسم الأيمن) قبل أن يتم تعريفه (في القسم الأيسر) » .

الإنطلاق من النتيجة

إن فكرة الإنطلاق من تعريف النتيجة الواجب الحصول عليها (إنها فكرة مشتركة مع كثير من طرق التحليل فمثلاً طريقة وارنيه Warnier تنطلق من وصف شكل السجل الحاوي على النتيجة) ليست إلا تطبيق للطريقة الديكارتية المعتمدة على تقسيم كل مسألة إلى عدة مسائل ثانوية مستقلة عن بعضها البعض والتي بدورها نجزّئها حتى الحصول على مسائل سهلة المعالجة .

ما نفعله هنا هو تعريف معرًف ما بواسطة معرِّفين آخرين والتي تكون تعريفاتهم مستقلة عن بعضها : نعطي إسماً لكل مسألة ثانوية للإهتمام بها وحدها فيما بعد. لن نهتم بأمر ترتيب تنفيذ الحسابات قياس الوصول الى مرحلة التنظيم .

ملاحظة : إن « النتيجة » و« المعطية » هما معرّفان محددان مسبقاً ويسمحان بالإتصال مع خارج البرنامج فمثلاً « النتيجة » يمكن أن تكون الشاشة (أو اللائحة) ، والمعطيّة يمكن أن تكون الملامس Clavier (أو البطاقات) :



2.1 ـ البرمجة في لغة الباسكال

إن التحليل الذي أجريناه يتطابق مباشرة مع برنامج باسكال

1 : يُكتب عنوان برنامج :

برنامج معرًف (دَخْل ، خَرْج) ؛ **Program** identificateur (input , ontput) ;

{ ملاحظة تحدد هدف البرنامج } .

حيث أن المعرًف المختار هو غير ذي معنى بالنسبة لباقي البرنامج .

2 : يترجم المعجم بمجموعة من التصريحات تبدأ بالكلمة ـ المفتاح **Var**

Var معرًف : النوع ؛ { ملاحظة } **Var** identificateur: type; / Commentaire

. . . . .

معرِّف : النوع ؛ { ملاحظة }

. . . . .

تُترجم الأنواع صحيح ، حقيقي ، سمة على التوالي بـ **char, real, integer** (كما نلاحظ فإن نَحْوَ لغة الباسكال هو إنكليزي) .

3 : تُترجم التعريفات الواحدة تلو الأخرى حَسَبَ الترتيب المُتَّبع بواسطة عبارات ؛ يبدأ هذا القسم بكلمة «begin» وينتهي بكلمة «end» .

- تعريف سهل مثل a = b + c يُكتب على الشكل التالي a: = b + c
- تعريف لنتيجة مثل : نتيجة = اكتب x يُكتب : writeln (x)
- تعريف لمعطيات مثل v, u, t « وفقاً » لمعطيّة يكتب read (t, u, v)
- عبارتين متتاليتين يتم فصلهما بواسطة نقطة ـ فاصلة .

إن كتابة برنامج باسكال مسألة سلسلة كفاية بحيث : يمكن إضافة تباعدات (إلاّ في المعرِّف أو في رمز للغة) ، يمكن إزالتها (إلاّ في حالات الإلتباس ، مثلاً بين begin و a : = . . . يجب وجود تباعد واحد على الأقل) ، إضافة أو إزالة قفزات على السطر . ما يجب تأمينه هو قراءة ممتازة للبرنامج .

مثال : الميزان

```
en-tête        | program balance(input,output);
               | { فوترة على ميزان مسجّل }
déclarations   | var prix:integer;          {الثمن المتوجب
               |                             دفعه من الزبون
               |                             بالقروش }
               |     poids:integer;          { الوزن بالغرام }
               |     prixUnitaire:integer;   { الثمن بالقروش
               |                             للكيلوغرام }
énoncés        | begin read(poids,prixUnitaire);
               |       prix:=poids*prixUnitaire div 1000;
               |       writeln(prix)
               | end.
```

(en-tête : عنوان ؛ Prix : الثمن ، Prix unitaire : سعر إفرادي ، déclarations : تصريحات ، Poids : الوزن ، énoncés : عبارات ، balance : ميزان)

التنفيذ

يتطلب هذا البرنامج معطيتين صحيحتين ويكتب نتيجة صحيحة : إذا أعطيناه 500 ، 2000 فإنه يطبع النتيجة 1000 .

يجري إدخال المعطيات تبعاً للترتيب المطلوب في البرنامج (في حالتنا هذه فإن الوزن يساوي 500 غرام والسعر الإفرادي يساوي 2000 قرش / كلغ . هاتين المعطيتين هما مفصولتين عن بعضهما بواسطة عدد من التباعدات أو برجوع إلى السطر ، دون تحديد للوحدة . كذلك فإن النتيجة تُكتب على شكل قيمة (1000 قرش)[1] .

---

(1) في بعض الحاسبات الآلية ، تكون السمة الأولى لكل سطر مطلوب طباعته ، سمة تحكّم لتقديم الورق ؛ وبذلك فإننا سنكتب writeln (' ', Prix)

ملاحظة : تبعاً لنوعية الحاسب الآلي المستعمل ، فإن التمثيل الشكلي للأحرف في البرنامج يمكن أن يتغير دون تشويه للمعنى :

Prix يمكن أن تكتب Prix, PRIX, **Prix,** Prix, prix إلخ . .
Pesé يمكن أن تكتب **PESE,** pese, pesé ، . . .
إلخ . . (أنظر 2.1) .

التعابير (Expressions)

يتألف التعبير في لغة الباسكال من :

● متغيّرات ممثلة بواسطة معرِّفين ؛ بكل معرِّف يتم ربط نوع (integer مثلاً) وفي كل لحظة ، قيمة مأخوذة في النوع .

مع **Var i** : integer ؛ فإن العبارة i := i + 1 تعني « خذ قيمة المتغير i (صحيحة) ، أضف اليها القيمة (الصحيحة) 1 ، هذا ما يعطي القيمة الجديدة (الصحيحة) للمتغير i » ؛

● ثوابت ؛

● مؤثّرات (Opérateurs) :

- على مُتأثرَيْن (Opérande) من النوع integer (صحيح)، تعطي المؤثرات + ، − ، * (الضرب) ، **div** (قسمة صحيحة) ، **mod** (حاصل قسمة صحيحة) جميعها نتيجة من النوع integer .
- على متأثرَيْن من النوع real (حقيقي) : + ، − ، * ، / (قسمة حقيقية) يعطوا نتيجة من النوع real .
- على متأثر من كل نوع integer أو real تعطي المؤثرات + ، − ، * ، / نتيجة من النوع real(جرى تحويل المتأثر integer إلى النوع real قبل العملية الحسابية) .
- / هي دائماً عملية حسابية على أعداد حقيقية .

فمثلاً 7 **div** 3 تساوي 2 و 7 **mod** 3 تساوي 1 (صحيح)
3 / 7.0 ، 7/3.0 ، 7/ 3 ، 3.0 / 7.0 يساووا 2.5 (حقيقي)

إن تقييم التعبير يتم من الشمال الى اليمين :
2 — 3 — 7 تعني 2 — (3 — 7)

لكن المؤثّرات الضاربة (* ، / ، **div** ، **mod**) لها الأسبقيّة على المؤثّرات الجامعة (− ، +) :
2 * 4 + 3 * 7 تعني (2 * 4) + (3 * 7)

يمكن دائماً تحديد ترتيب التقييم للتعبير وذلك بالوضع بين قوسين :
2.5 / 3.0 / 7.5 تعني 2.5 / (3.0 / 7.5) أي (2.5 * 3.0) / 7.5 .

التعيين (Affectation)

لا يمكن التعيين لمتغير صحيح إلاّ قيمة صحيحة :

a: = a / 2 هي غير قانونية (a / 2 هي حقيقية)

لا يمكن التعيين لمتغير حقيقي إلا قيمة حقيقية أو قيمة صحيحة (فإذن هناك تحويل أوتوماتي سوف يتم) :

X : = 7 **div** 3 تعيّـن للمتغير الحقيقي X القيمة 2.0

سوف يتم تفصيل هذه القواعد في الفصل الثاني) .

3.1 ـ التكرارية مع عدّاد (Iteration avec Compteur)

يتم الحساب في كثير من الحالات ، على متسلسلات من القيم وليس فقط على قيم بسيطة .

من الناحية العامة ، نستعمل المتسلسلة (Suite) إمّا لمعرفة كل حدودها ، مثلاً اكتب متسلسلة من الفواتير ، إمّا لمعرفة الحدّ الأخير ، مثلاً اكتب المبلغ المتوجب دفعه بعد عدة وزنات . يتم الحصول على عناصر المتسلسلة بواسطة التكرارية ؛ يُحسَبْ الحد ذي الدليل i من متسلسلة :

ـ إمّا بشكل مستقل عن الحدود الأخرى = $u_i = f(k)$

(مثال : اكتب متسلسلة من الفواتير ، الناتجة عن وزنات مستقلة)

ـ إمّا تبعاً للدليل الخاص به : $u_i = f(i)$

(مثال : اكتب جدولاً للضرب بـ N : $t_i = N * i$)

ـ إمّا تبعاً للحد السابق : $u_i = f(u_{i-1})$

(مثال : المبلغ الإجمالي المتوجب دفعه : $S_i = S_{i-1} + Prix_i$)

أخيراً يمكن أن يكون عدد الحدود معروفاً صراحة (اكتب 20 فاتورة) ، أو محدَّد بخاصة (نوقف جمع حدود المتسلسلة عندما يصبح الباقي أقل من قيمة معطيّـه) . تؤدي الحالة الأولى الى تكرارية مع عدّاد :

$$u = \textbf{dernier } u_i = f(k, i, u_{i-1}) \textbf{ pour } i \textbf{ de } d \textbf{ à } a$$

(dernier = آخر، pour = لِـ « الحلقة لِـ »)

بينما الحالة الثانية تؤدي إلى تكرارية مشفوعةِ بشريط توقف :

$$u = \textbf{dernier } u_i = f(k, i, u_{i-1}) \textbf{ pour } i \textbf{ de } d \textbf{ inclus } \textit{arrêt}$$

(arrêt = توقف ، inclus = مضمَّـن)
إذا كان الحدّ الأخير مضمَّـن في الحساب

$$u \;=\; \textbf{dernier}\; u_i \;=\; f\,(k,\, i,\, u_{i-1})\; \textbf{pour}\; i\; \textbf{de}\; d\; \textbf{tant que}\; non\; arrêt$$

(tant que = طالما « الحلقة طالما »)
إذا كان الحدّ الأخير مستثنى في الحساب

التكرارية للكتابة

تؤدي المسألة « إجراء n فوترة على ميزان مسجِّـل » الى المواصفة « إحسب الثمن المتوجب دفعه (عدد صحيح من القروش) بعد كلٍّ من الـ n وزنات ؛ الثمن هو . . . (أنظر 1.1) المطلوب إذن هو كتابة.نتيجة سهلة لعدد n من المرات

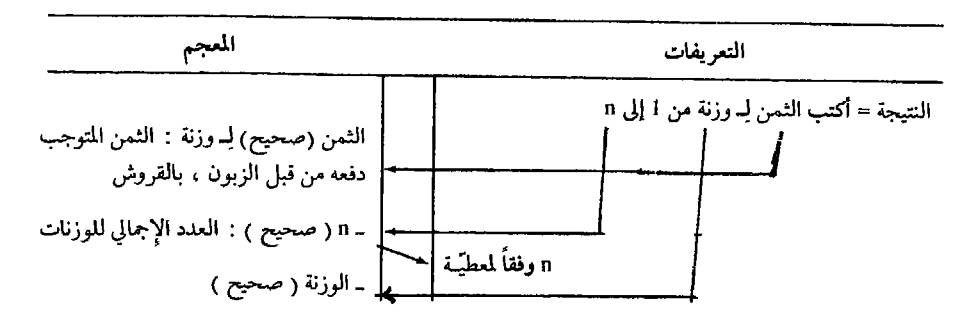
résultat* = écrire *prix* pour *pesée* de *1* à *n

(résultat = نتيجة ، écrire = اكتب ، prix = ثمن ، pesée = وزنة)

رُقِّـمت الوزنات من 1 إلى n ؛ يجب أن يعاد حساب الثمن عند كل وزنة ، فإذن لا يجب أن يظهر تعريفه في نفس مستوى تعريف النتيجة : لذلك ندخل جدولاً ثانوياً لكل تعريف مكرَّر . في المعجم : نسجِّـل بأنه يجب إدخال جدولاً ثانوياً لتعريف الثمن وذلك عن طريق كتابة :

الثمن المتوجب دفعه من قبل الزبون ، بالقروش
(prix (entier) **pour** pesée الثمن (صحيح) لِـ وزنة)

في المقابل ، فإن تعريف n ليس له علاقة بالوزنة : نعمل على إظهار تعريفـه في الجدول الأساسي . المعرِّف « وزنة Pesée » سيـوضـع فـي المعجـم ، وهو مشطوب مسبقاً لأنه سبق تعريفه (إنه يساوي 1 ، ومن ثم 2 ، . . . وأخيراً n) .



(كان بمقدورنا تماماً إعطاء قيمة لـ n : n = 10 . يعتمد هذا الخيار على المحتوى الحقيقي للمسألة)

بعد أن إكتمل الجدول الأساسي ، فإننا سنكتب الجداول الثانوية ، وذلك بالتجميع في كل منها لكل المعرِّفين الذين لديهم نفس حقل التعريف .

سيكون عنوان الجدول الثانوي :

حقل التعريف ← المعرفين الذي يجب تحديدهم

(في حالتنا : لِـ وزنة ← ثمن) .

التعريفات		المعجم	
① النتيجة = أكتب الثمن لـ وزنة من 1 إلى n	2	ـ الثمن (صحيح) لـ وزنة : الثمن المتوجب دفعه من قبل الزبون ، بالقروش	②
		n (صحيح) : العدد الإجمالي للوزنات	②
③ n وفقاً لمعطيّة	1	ـ الوزنة (صحيح)	②
④ الثمن = الوزن * السعر الإفرادي 1000 div	2	لـ وزنة ← ثمن	
		الوزن (صحيح) : الوزن بالغرام	⑤
⑥ الوزن ، السعر الإفرادي وفقاً لمعطيّة	1	السعر الإفرادي (صحيح) : السعر بالقروش للكيلوغرام	⑤

ملاحظة : لكي نكون فعلياً دقيقين ، يجب تذييل الثمن ، الوزن والسعر الإفرادي للوزنة بأدلّة ، لأنه توجد قيمة في كل خطوة من التكرارية .

تتم عملية ترتيب التعريفات على كل جدول .

في لغة الباسكال ، فإن التعريف :

résultat* = écrire *x* pour *y* de *z* à *u

يُكتب على الشكل التالي :

for *y*: = *z* **to** *u* **do begin**

تعريفات تكرارية }
traduction de la table «**pour** y» (ترجمة الجدول « لِـ y »)
writeln (x)

end

(إذا كان الجدول « pour y » فارغاً ، فإنه يمكن إهمال begin وend)
يصبح البرنامج في حالنا هذه :

```
program balance2(input,output);
{ فوترة عدة وزنات على ميزان مسجِّل }

var prix:integer;        { لكل وزنة ، الثمن المتوجب دفعه من قبل
                             الزبون ، بالقروش}
    n:integer;           { العدد الإجمالي للوزنات }
    pesee:integer;
    poids:integer;       { لكل وزنة ، الوزن بالغرام }

    prixUnitaire:integer; {لكل وزنة ، الثمن بالقروش للكيلوغرام}

1 → begin read(n);
2 →        for pesee:=1 to n do begin
sous- | 1 →    read(poids,prixUnitaire);
table | 2 →    prix:=poids*prixUnitaire div 1000;
               writeln(prix)
           end
    end.
```

(sous-table : جدول ثانوي)

عند التنفيذ ، يستلزم هذا البرنامج أولاً قيمة n ، ثم n مرة قيمة الوزن (أي الوزنة) والسعر الإفرادي (للوزنة) . إذا أعطيناه :

3 500 2000 100 5000 200 1000

يكتب في النهاية 1000

500

200

إرجاع الثوابت

إن تحليل المسألة « إجراء n فوترة على ميزان مسجِّل لسلعة واحدة » ، وبنفس الطريقة السابقة ، يؤدي الى إمداد البرنامج عند التنفيذ ، n مرة نفس القيمة للسعر الإفرادي : « السعر الإفرادي وفقاً لمعطيّة » هو ثابت في التكرارية . من المهم إذن إخراج هذا التعريف من الجدول الثانوي أي إرجاعه إلى الجدول الأساسي :

المعجم		التعريفات
ـ الثمن (صحيح) لوزنة	3	النتيجة = أكتب الثمن لـ وزنة من 1 إلى n
ـ n (صحيح) ـ وزنة (صحيح)	1 2	n وفقاً لمعطيّة السعر الإفرادي وفقاً لمعطية
لـ وزنة ← سعر ـ الوزن (صحيح)	2	الثمن = الوزن * السعر الإفرادي 1000 div
ـ السعر الإفرادي (صحيح)	1	الوزن وفقاً لمعطيّة

هذا ما يغيّر البرنامج :

```
...
begin read(n,prixUnitaire);
      for pesee:=1 to n do begin
          read(poids);
          prix:=poids*prixUnitaire div 1000;
          writeln(prix)
      end
end.
```

ملاحظة : read (x, y, ..) تعني read (x) ؛ read (y) ؛ . . .

عند التنفيذ ، إذا كانت المعطيات المقدّمة هي :

4 2000 500 200 800 1000

فإن النتيجة المكتوبة تصبح

1000
400
1600
2000

مثال : إطبع مُربَّع الأعداد

المسألة : إطبع جدولاً لمربّـع أوّل 25 عدداً صحيحاً .

التحليل

المعجم		التعريفات
- n (صحيح) - مربّـع (صحيح) لِـ n : n * n	1	النتيجة = اكتب n ، مربّـع لِـ n من 1 إلى 25
لِـ n ← مربّـع	1	مربّـع = n * n

هنا يتعلق المعرِّف المحدَّد بالتكرارية (مربّـع ذي الدليل n) صراحة بدليله ؛ ولا يوجد معطيات (العدد 25 تم تحديده في نص المسألة) . فيما عدا هذه التفاصيل ، فإن التحليل هو نفسه الذي سبق .

البرنامج :

```
program editerCarres(input,output);
{ إطبع جدولاً لمربّـع أول 25 عدداً صحيحاً }
var n: integer;
    carre: integer;  {n*n}
begin
      for n:=1 to 25 do begin
         carre:=n*n;
         writeln(n,carre)
      end
end. (1)
```

carré= مربّع ، éditer = إطبع ، table = جدول ، entier = صحيح .

ملاحظة : إن جدولة f(x) من أجل x تتغير من A إلى B (أعداد صحيحة) وبخطوات تساوي 1 ، تؤدي إلى برنامج شبيه جداً بهذا البرنامج : نستبدل n * n بـ f(x) .

تكرارية للحساب

لنفرض المسألة « إحسب المبلغ الإجمالي المتوجب دفعه ، بعد عدة وزنات على ميزان مسجِّل » . لم يعد المقصود كتابة نتيجة عند كل وزنة ، بل كتابة المبلغ الإجمالي؛ عند كل وزنة جديدة ، يكون المبلغ الإجمالي الجديد هو حاصل جمع المبلغ الإجمالي القديم مع ثمن الوزنة :

(1) في حال كانت طباعة الأرقام الصحيحة تتم بدون تباعد ، فإنه يجب كتابة :
Writeln (n, '   ', Carré)

$$total_{pesée} = \text{total}_{pesée-1} + \text{prix}_{pesée}$$

(مجموع الوزنة = مجموع الوزنة - 1 + ثمن الوزنة)

والنتيجة النهائية المطلوب كتابتها هي المجموع الأخير :

النتيجة = أكتب مجموع

مجموع = آخر مجموع = مجموع وزنة = مجموع وزنة - 1 + ثمن وزنة لِـ وزنة من 1 إلى n

الأدلّـة هي غـير ضرورية : يكفي وجود إمكـانية تفـريق الحدّ الحـالي من الحدّ السابق . من المتفق عليه ، في تعريف ، وضع خط فوق الحد السابق :

مجموع = آخر مجموع = مجموع + ثمن لـ وزنة من 1 إلى n

تُعـرَّف المتسلسلة « مجمـوع » بـالتثنيـة إلى الـوراء récurrence (نحسب حـدود المتسلسلة بواسطة علاقة تثنية إلى الوراء (1) ، مثل $u_i = f(u_{i-1})$. لا يمكن إتمام مثل تلك العملية إلاّ إذا كنا نعرف الحدّ الأول : مجموع $_0$ = 0 .

الدليل غيرضروري: يكفي معرفة الحد الأول، هذا ما نسجّـله : أوَّل مجموع = 0 .

التعريفات		المعجم
نتيجة = أكتب مجموع مجموع = آخر مجموع = مجموع + ثمن لِـ وزنة من 1 إلى n أول مجموع = 0 n وفقاً لمعطية	4 3 2 1	ـ مجموع (صحيح) : متسلسلة المجاميع الجزئية ، وزنة تلو وزنة ـ n (صحيح) : عدد الوزنات الثمن (صحيح) ــــ لـ وزنة : متسلسلة الأثمان الواجب دفعها ، بالقروش ـ الوزنة (صحيح)
		لـ وزنة ← ثمن
الثمن = الوزن * السعر الإفرادي 1000 div	2	ـ الوزن (صحيح) : الوزن بالغرام
الوزن ، السعر الإفرادي وفقاً لمعطيّـة	1	ـ السعر الإفرادي (صحيح) : الثمن بالقروش 1 للكيلوغرام

(1) عن طريق تعريف المتسلسلة بالتثنية الى الوراء : $\begin{cases} u_i = f(u_{i-1}) \\ u_0 = a \end{cases}$

يمكن حساب $u_1 = f(a)$, $u_2 = f(u_1) = f(f(a))$ ، إلخ . .

لـ { مجموع$_i$ = مجموع$_{i-1}$ + ثمن$_i$ ، مجموع$_0$ } نحصل على مجموع$_i$ = ثمن$_i$ + ثمن$_{i-1}$ + ثمن$_{i-2}$ + . . . + ثمن$_1$

أي مجموع$_i$ = $\sum_{j=1}^{i}$ ثمن

تكتب التعريفات

v = **dernier** $v = f(\bar{v})$ **pour** *i* **de** *départ* **à** *arrivée*
premier v = *expression*

(= expression ، أول = premier ، وصول = arrivée ، انطلاق = départ ، لِـ = pour ، آخر = dernier
تعبير ، جدول = table ، ترجمة = traduction)

في لغة الباسكال

v: = *expression;*
for *i*:= *départ* **to** *arrivée* **do begin**
  traduction de la table « **pour** *i* »
  v: = *f (v)*
end

بالفعل ، فإن المعرّف الموضوع في القسم الأيسر من تعيين (affectation) يتطابق مع « القيمة الجديدة » ، بينما يتعلق الأمر في القسم الأيمن بالقيمة القديمة :

$$x = \bar{x} + \ldots \quad \Rightarrow \quad x := x + \ldots$$

التحليل ⇒ باسكال

البرنامج

```
program balance3(input,output);
{calculer le total à payer après plusieurs pesées
 sur une balance enregistreuse}
var total:integer;        { احسب المبلغ الإجمالي المتوجب دفعه بعد عدة وزنات على ميزان مسجّل }
                          { متسلسلة المبالغ الإجمالية الجزئية وزنة بعد وزنة }
    n:integer;            { عدد الوزنات }
    prix:integer;
                          { لكل وزنة ، الثمن المتوجب دفعه بالقروش }
    pesee:integer;
    poids:integer;        { لكل وزنة ، بالغرام }
    prixUnitaire:integer; { لكل وزنة ، الثمن بالقروش للكيلو }
```

```
1,2 →                 begin
3 →                     read(n); total:=0;
                        for pesee:=1 to n do begin
جدول  | 1 →                 read(poids,prixUnitaire);
ثانوي | 2 →                 prix:=poids*prixUnitaire div 1000;
   تعريف تكراري →           total:=total+prix
                        end;
4 →                     writeln(total)
                      end.
```

تعطي المعطيات 2500 2000 5000 1000 2 النتيجة 10 000

مثال : وسط حسابي (moyenne)

المسألة : احسب الوسط الحسابي لـ n قيمة

المواصفة : احسب الوسط الحسابي لـ n قيمة (n معطيّـة) :

وسط حسابي = (مجموع الـ n قيمة) / n ؛ القيم هي حقيقية ،

(مجموع الـ i قيمة) = (مجموع الـ 1 − i قيمة) + (القيمة ذات الترتيب i)

التحليل :

	التعريفات		المعجم	
①	نتيجة = أكتب وسط حسابي	5	ـ وسط حسابي (حقيقي) : وسط حسابي	②–
③	سط حسابي = مجموع / n	4	ـ مجموع (حقيقي): مجموع الـn قيمة	④–
⑤	مجموع = آخر مجموع = مجموع + ثمن لِـ عدد من 1 إلى n	3	n (صحيح): عدد القيم	④–
⑤	أول مجموع = 0	2	v (حقيقي) لِـ عدد: قيمة كل عدد	⑥–
⑦	n وفقاً لمعطيّـة	*1*	ـ عدد (صحيح)	⑥–

⑧ v وفقاً لمعطيّـة | 1 | لِـ عدد ← v

⑨

البرنامج :

```
program moyenne(input,output);
{moyenne de n valeurs réelles} { معدّل n قيمة حقيقية }
var moyenne:real;              { وسط حسابي لـ n قيمة }
    somme:real;                { مجموع الـ n قيمة }
    n:integer;       { عدد القيم }
    v:real;          { رقم القيمة }    { لكل عدد ، قيمته }
    nombre:integer;
begin
  read(n); somme:=0;
  for nombre:=1 to n do begin
    read(v);
    somme:=somme+v
  end;
  moyenne:=somme/n;
  writeln(moyenne)
end.
```

moyenne = وسط حسابي ؛ valeur = قيمة ؛ somme = مجموع ؛ nombre = عدد

أمثلة من المعطيات : 0.2 −   0.06   3.14   17.0 −   4   النتيجة : 3.5 −

التكرير للكتابة والحساب

إن المسألة « إحسب المبلغ الإجمالي المتوجب دفعه ، بعد عـدة وزنات عـلى ميزان مسجّـل » تؤدي إلى تعريف نتيجتين ؛ من جهة المجموع النهائي ومن جهة أخرى الثمن الحاصل عن كل وزنة :

نتيجة = نتيجة 1 ، نتيجة 2

نتيجة 1 = أكتب ثمن لِـ وزنة من 1 إلى n

نتيجة 2 = أكتب مجموع

سيكون في المعجم ، للنتيجة 1 والنتيجة 2 النوع نص (Text) ، ذلك لكونها قيم خارجية وليست داخلية (مثل حقيقي وصحيح) . إن التعريفـات نتيجة 1 ومجمـوع ، لهم نفس حقل التعريف (لِـ وزنة من 1 إلى n) : سيتم دمجهم في البرنامج .

التعريفات		المعجم	
① نتيجة = نتيجة 1 ، نتيجة 2	5	- نتيجة 1 (نص)	—②
③ نتيجة 1 = أكتب ثمن لِـ وزنة من 1 إلى n	3	- نتيجة 2 (نص)	—②
⑤ نتيجة 2 = أكتب مجموع	4	- الثمن (صحيح) لِـ وزنة: متسلسلة الأثمان الواجب دفعها، بالقروش	—④
⑦ n وفقاً لمعطية	2	- وزنة (صحيح)	—④
⑧ مجموع = آخر مجموع = مجموع + ثمن لـ وزنة من 1 إلى n	3	- n (صحيح): عدد الوزنات	—④
⑧ أول مجموع = 0	1	- مجموع (صحيح): متسلسلة المجاميع - الجزئية ، وزنة تلو وزنة	—⑥
		لِـ وزنة ← ثمن ⑨	
⑨ الثمن = الوزن * السعر الإفرادي div 1000	2	- الوزن (صحيح) : الوزن بالغرام	—⑩
⑪ الوزن ، السعر الإفرادي وفقاً لمعطيـة	1	- السعر الإفرادي (صحيح) : الثمن بالقروش للكيلوغرام	—⑩
	⑫		

بذلك يصبح قسم العبارات في البرنامج :

```
begin
  read(n); total:=0;
  for pesee:=1 to n do begin
    read(poids,prixUnitaire);
    prix:=poids*prixUnitaire div 1000;
    writeln(prix);
    total:=total+prix
  end;
  writeln(total)
end.
```

لا يظهر المعرِّفان نتيجة 1 ونتيجة 2 في البرنامج ، إنهما يتطابقان مع قسمة منطقية للائحة النتائج الغير موجودة في لغة الباسكال .

إذا كانت معطيات البرنامج هي 1000 200 5000 100 2000 500 3

النتيجة تصبح 1000
500
200
1700

4.1 ـ شَرْطيّ

عندما تتعلق طريقة حساب قيمة بشرط ، نستعمل تعريفاً شرطيّـاً :

identificateur = *expression*$_1$ **si** *condition*, *expression*$_2$ **sinon**

(معرِّف = تعبير 1 إذا شرط ، تعبير 2 وإلاّ)

إذا تحقق الشرط ، فإن تعريف المعرِّف يصبح : معرِّف = تعبير 1 ؛ إذا لم يتحقق الشرط يصبح عندنا : معرِّف = تعبير 2

هذه هي حالة حساب الساعات الإضافية لعامل بالساعة . نفترض بأن الساعات الإضافية (أكثر من 39 ساعة في الأسبوع) تدفع بنسبة 125% من القيمة الأساسية للساعة ، وبأن حسماً يجري تطبيقه على مجمل الأجر الخام قيمته 4.75% .

التعريفات		المعجم	
① نتيجة = أكتب الأجر الصافي	6	ـ الأجر الصافي (حقيقي): الأجر الأسبوعي المستحق	② –
③ الأجر الصافي = الأجر الخام – الحسم	5	ـ الأجر الخام (حقيقي): الأجر قبل الحسم	④ –
⑤ الأجر الخام = الساعات * سعر إذا الساعات = <39 ، 39 * سعر + (الساعات – 39) * سعر * 1.25 وإلاّ	3	ـ الحسم (حقيقي) : 4.75%	④ –
⑦ الحسم = الأجر الخام * 0.0475	4	ـ الساعات (حقيقي): عدد الساعات المتمّمة	⑥ –
⑧ الساعات وفقاً لمعطيّـة	2	ـ سعر (حقيقي) : أجر الساعة	⑥ –
⑨ سعر وفقاً لمعطيّـة	1		

⑩

يترجم الشرط بواسطة مؤثرات مقارنة :

التنويط المتّبع	≥	>	≤	<	≠	=
التنويط في لغة الباسكال	>=	>	<=	<	<>	=

في لغة الباسكال ، يترجم التعريف الشرطي **x = y si c, z sinon** بالعبارة :

```
if c then begin
  traduction de la table « si c »
  traduction de « x = y »
  end
else begin
  traduction de la table « sinon c »
  traduction de « x = z »
  end
```

(traduction = ترجمة ، table = جدول ، si = إذا ، sinon = وإلاّ)

ملاحظة : إذا وُجدت عبارة واحدة في الشعبة (branche) (then أو else) ، فإنه لا يجب إحاطتها بـ begin وend .

البرنامج :

```
program paye(input,output);

var salaireNet:real;      { الدفع الأسبوعي لعامل بالساعة }
                          { الأجر الأسبوعي المتوجب }
    salaireBrut:real;     { الأجر قبل الحسم }
    retenue:real;         { 4.75% من الخام }
    heures:real;          { عدد الساعات المشغولة }
    taux:real;            { الأجر بالساعة }
begin
  read(taux,heures);
  if heures<=39 then
    salaireBrut:=heures*taux
  else
    salaireBrut:=39*taux+(heures-39)*taux*1.25;
  retenue:=salaireBrut*0.0475;
  salaireNet:=salaireBrut-retenue;
  writeln(salaireNet)
end.
```

(Paye = الدفع ؛ salaire = أجر ؛ retenue = الحسم ؛ brut = خام ؛ Net = صافي ؛ taux = سعر ، heure = ساعة)

معطيات : 30,0 28.53 النتيجة 815.24

معطيات : 59.0 28.53 النتيجة 1739.19

مثال : معادلة من الدرجة الثانية

حلّ معادلة من الدرجة الثانية $ax^2 + bx + c = 0$ ، حيث $a = 0$

بعد مراجعة كتابات متخصصة ، يتبيّن بأن لهذه المعادلة جذرين حقيقيين x_1 وx_2 إذا كانت الكمية $d = b^2 - 4ac$ هي موجبة أو تساوي صفر ، ولها جذرين عُقدِيَّين (Complex) R + iS وR − iS إذا كانت d سالبة :

résultat = **écrire** *x1, x2* **si** *d* > = *0*, **écrire** *R*, ' + *i*', *S*, *R*, ' − *i*', *S* **sinon**

(écrire = أكتب ، résultat = نتيجة ، si = إذا ، sinon وإلاّ)

إن التنويط '+i' يعنى بأنه يجب كتابة السمتين '+' و'i' كما هما . إن شكل النتيجة سيكون : 2.0 − i4.5 2.0 + i4.5 ؛ إن سلسال السمات المنوّط بين علامات حذف يكتب كما هو .

سيتم كتابة x1 وx2 في المعجم مع حقل تعريف 'Si d > = 0' ؛ R وS مع الحقل 'Sinon d > = 0'

المعجم		التعريفات
② جذر حقيقي : si d > = 0 (حقيقي) x1 ـ	3	① نتيجة = أكتب 'جذرين حقيقيين :' ، x1 ، 'و' x2 إذا d > = 0 ، أكتب 'جذرين عُقديّين :' ، R ، '+' ، s ، 'i' و R ، '−' ، s ، 'i' وإلا
② جذر حقيقي : si d > = 0 (حقيقي) x2 ـ		
② القسم الحقيقي : sinon d > = 0 (حقيقي) R ـ من الجذور العُقدية	2 1	③ d = b * b − 4 * a * c ⑤ a, b, c وفقاً لمعطية
② القسم التخيُّلي من الجذور العُقدية : sinon d > = 0 (حقيقي) S ـ		
② (discriminant) مُميِّز : (حقيقي) d ـ		
④ مُعامل لـ x^2 : (حقيقي) a ـ		
④ مُعامل لـ x^1 : (حقيقي) a ـ		
معامل لـ x^0 : (حقيقي) c ـ		
si *d* > = *0* → *x1, x2*	1 2	$x1 = (-b + \sqrt{d})/(2 * a)$ ⑥ $x2 = (-b - \sqrt{d})/(2 * a)$ ⑦
sinon *d* > = *0* → *R, S*	1 2 ⑩	$R = -b/(2 * a)$ ⑧ $S = \sqrt{-d}/(2 * a)$ ⑨

تنوط العملية $\sqrt{d}$ في لغة الباسكال كما يلي : Sqrt (d)

البرنامج

```
program secondDegre(input,output);
 { ax² + bx + c = 0 حل }
var x1,x2:real;      { إذا d > = 0 جذور حقيقية }
    R,S:real;                  { إذا d < 0 ، الأقسام الحقيقية والتخيّلية للجذور العقدية }

    d:real;          { مميّز }
    a,b,c:real;      { مُعاملات }
begin
  read(a,b,c);
  d:=b*b-4*a*c;
  if d>=0 then begin    { جذرين عقديين }
    x1:=(-b+sqrt(d))/(2*a);
    x2:=(-b-sqrt(d))/(2*a);
    writeln('deux racines reelles:');
    writeln(x1,' et',x2)
    end
  else begin            { جذرين حقيقيين }
    R:=-b/(2*a);
    S:=sqrt(-d)/(2*a);
    writeln('deux racines complexes:');
    writeln(R,'+',S,'i  et',R,'-',S,'i')
  end
end.
```

(second degré = الدرجة الثانية ، racine = جذر ، complexe = عقدي ، réelle = حقيقي) .

معطيات : 4.0   − 4.0   2.0

النتيجة : جذرين عُقديين 1.0i + 1.0 و1.0i − 1.0

معطيّـات : 0.0   1.5   1.0

النتيجة : جذرين حقيقيين 0.0 و1.5 −

الشرطيّـة والتكراريّـة

في متسلسلة من 10 معطيّـات صحيحة ، نودّ عدّ تلك التي هي موجبة أو مساوية لصفر ، وكذلك تلك السالبة .

المعجم		التعريفات
- pos (صحيح) : عدد المعطيات الموجبة أو المساوية لصفر	3	نتيجة = أكتب 'موجبة = ' ، pos ، 'سالبة' ، neg
- neg (صحيح) : عدد المعطيات السالبة "	2	pos = آخر pos = pos + x لِـ i من 1 إلى 10
- x (صحيح) لِـ i:1 إذا كانت المعطيّة موجبة أو مساوية لصفر	1	أول pos = 0
- i (صحيح)	3	neg = 10 - pos
لِـ i ← x		
- v (صحيح) : معطيّـة مقروءة	2 1	x = 1 si v > = 0, 0 sinon وفقاً لمعطيّـة v

البرنامج

```
program positif(input,output);
{ عد القيم الموجبة أو المساوية لصفر والقيم السالبة ، في متسلسلة من 10 معطيات }
                      { عدد المعطيات الـ 0 = < }
var pos:integer;
    neg:integer;      { عدد المعطيات الـ 0 > }
    x:integer;        { لكل معطية + 1 إذا موجبة ، وإلاّ 0 }

    i:integer;
    v:integer;        { لكل معطية = القيمة المقروءة }
begin
  pos:=0;
  for i:=1 to 10 do begin
    read(v);
    if v>=0 then x:=1 else x:=0;
    pos:=pos+x
  end;
  neg:=10-pos;
  writeln('positives=',pos,' negatives=',neg)
end.
```

(سالب = négatives ، موجب = positives)

معطيّـات : 111 0 4 − 1 2 0 3 4 − 7 − 1

نتيجة : موجب = 7 ، سالب = 3

5.1 - تكراريّـة مع توقف

إن توقف التكراريّـة يمكن أن يحدد :

- صراحةً : id = تعريف لِـ i من « إنطلاق » إلى « وصول » إذا عرفنا مسبقاً عدد خطوات التكرارية .

ـ بواسطة شرط ؛ في هذه الحالة فإن خطوات التكرارية تتم عندما :

ـ يتحقق الشرط :

id = تعريف لـ i من « إنطلاق » طالما « شرط »

إن آخر قيمة ناتجة عن هذه العملية ، أي أوّل قيمة غير محقّقة للشرط ، سوف تُقصى عن التكرارية (EXCLUE)

ـ الشرط غير متوافر

id = تعريف لـ i من « إنطلاق » متضمّـناً (**inclus**) « شرط »

إن آخر قيمة ناتجة هي داخلة في التكرارية .

تكرارية مع إقصاء (أو إقتصار) (Itération avec exclusion)

إن إرادة الإقصاء من التكراريّـة ، للحدّ الأخير الناتج تعني بـأن المتاليـة المكرّرة سوف تنتهي بانتاج حدّ الذي رُبّـما إستُخدِمَ في الخطوة التالية :

إقصاء ← طالما ← **(tant que)**	إنتاج الحدّ الأوّلي ، من ثمّ إذا توفر الشرط العمل على ـ إستعمال الحدّ ـ إنتاج الحدّ التالي . ـ من ثمّ إعادة الكرّة

إن التعبير « إنتاج حدّ » يعني هنا حساب الحدّ الفعلي ، دليله في المتسلسلة ، وقيمة الشرط المتوافق ؛ عند الترجمة إلى الباسكال ، فإن ، حساب الشرط ، إذا كان سهلاً ، يمكن أن يتم في لحظة الإختبار ويمكن إلغاء عملية حساب الدليل إذا لم يكن الدليل مستعملاً صراحة .

معرّف = تعريف لِـ i من « إنطلاق » طالما « شرط »

يمكن إعادة كتابة تلك العبارة كما يلي :

i: = *départ;* (إنطلاق)
calcul du terme initial (d'indice *départ*) (حساب الحدّ الأوليّ)
c: = condition (sur le terme initial) (شرط على الحدّ الأوليّ)
while *c* **do begin**
(= تعريف(1)) traduction de l'utilisation du terme (ترجمة إستعمال الحد)
i: = *i* + *1;*
(= جدول ثانوي) traduction de la production du terme (ترجمة إنتاج الحدّ)
c: = condition (sur le terme produit) (شرط على الحدّ المنتج)
end

(1) إلا إذا كان المقصود هو تعريف مكرر ، الذي يؤخذ إذاً في هذه الحالة كقسم من الجدول الثانوي (إنتاج) .

(إن المعرّف الممثّل للشرط «C» هنا ، سيتم التصريح عنه في الباسكال مع النوع « البولي »)

مثال : مجموع

لنفرض المطلوب حساب مجموع عدة قيم صحيحة موجبة ؛ لا نعرف عددها ، لكنها متبوعة بالقيمة «1 −» .

يتعلق الأمر إذن بتكرارية مع شرط توقف ، الحدّ الأخير هو مقصى عن الحساب .

التعريفات		المعجم
نتيجة = أكتب مجموع	3	ـ مجموع (Somme) (صحيح)
مجموع = آخر مجموع = مجموع + v لِـ n من 1 طالما v < > 1 −	2	ـ V (صحيح) لـ n: متسلسلة من القيم
أوّل مجموع = 0	1	ـ n (صحيح)
v وفقاً لمعطيّة	1	لِـ n ← v

كون الشرط سهل التعبير ، فسيتم ترجمته مباشرة ؛ وبما أن الدليل غير ظاهر صراحة ، فسيتم إلغاءه :

البرنامج

```
program somme(input,output);
{ مجموع القيم المقروءة ، متبوعة بـ 1 − }
var somme:integer;
    v:integer;        { لكل معطيّة ، قيمتها }
begin
  somme:=0;
  read(v);
  while v<>-1 do begin
    somme:=somme+v;
    read(v)
  end;
  writeln(somme)
end.
```

تكرارية مع تضمين (Itération avec inclusion)

يتم إستعمال الحدّ الأخير الناتج :

تضمين ← متضمناً (inclus) ←	التدميث عند الإقتضاء ، من ثمّ - إنتاج حدّ - إستعماله - من ثم إعادة الكرة إذا لم يتوفر الشرط

معرِّف = تعريف لِـ i من إنطلاق متضمّـناً شرط

يمكن إعادة كتابة تلك العبارة كما يلي :

```
i:= départ - 1;
repeat
        i:= i+1;
        traduction de la table « pour i »    ( ترجمة الجدول لـ i )   ( = إنتاج )
        traduction de la définition          ( ترجمة التعريف )       ( = إستعمال )
        c:= condition                        ( شرط )
until c
```

حيث ان فتح المزدوجين begin-end لا لزوم له

ملاحظة : هذا الشكل من التكراريّـة يستعمل خاصة لجمع المتسلسلات .

مثال : نيوتن Newton

إن الجذر التربيعي لعدد موجب a هو أيضاً نهاية المتسلسلة المكرّرة $u_i = (u_{i-1} + u_{i-1}) / 2$ (صيغة نيوتن) .

بالفعل ، إذا سمّينا 1 نهاية المتسلسلة ، فإن 1 تخضع للمعادلة $l = (l + a/l)/2$ أي $l^2 = a$. لكي نحصل على l مع دقة إبسيلون (précision epsilon) معطيّـة ، يجب توقيف التكرّر للقيمة x للدليل بشكل أن $| a - U^2_x | / a$ تصبح أصغر من إبسيليون (القيمة الأولية u_0 هي غير ذات أهمّيـة كبرى) :

تعريفات		معجم
نتيجة = أكتب u	5	ـ u (حقيقي): متسلسلة التقريبات
u = آخر $u = (\bar{u} + a/\bar{u})/2$ لـ k من 1 متضمناً $\|a - u\|/a < epsilon$	4	ـ a (حقيقي) : العدد الذي نبحث عن جذره ـ k (صحيح) ـ ابسيلون (حقيقي) : دقة
أوّل u = 1	3	
a وفقاً لمعطيّـة	2	
ابسيلون = 0.00001	1	

تنوّط القيمة المطلقة في لغة الباسكال abs .

البرنامج

```
program Newton(input,output);
{ حساب الجذر التربيعي ، على طريقة نيوتن }
var u:real;    { متسلسلة التقريبات }
    a:real;    { العدد الذي نبحث عن جذره }
    epsilon:real;    { دقة }
begin
  epsilon:=0.0001; read(a); u:=1;
  repeat
    u:=(u+a/u)/2
  until (abs(a-u*u)/a)<epsilon;
  writeln(u)
end.
```

إذا أردنا الحصول في نفس الوقت على الجذر وعدد التكريرات ، فإنه يكفي تغيير التعريف 5 إلى أكتب u, k لكي يجب عندئذٍ وصف الحسابات المجراة على k في البرنامج :

```
begin
  epsilon:=0.0001; read(a); u:=1; k:=0;
  repeat
    k:=k+1;
    u:=(u+a/u)/2
  until (abs(a-u*u)/a)<epsilon;
  writeln(k,u)
end.
```

دون أن ننسى التصريح k : integer

التكريرات

	عدد الخطوات المعلومة		
	صراحة	بواسطة شرط	
		الحد الأخير ضمناً	الحدّ الأخير مقصى
عبارة صريحة	**pour** *i* **de** *d* **à** *a*	**pour** *i* **de** *d* **inclus** *c (i)*	**pour** *i* **de** *d* **tant que** *c (i)*
باسكال	**for**	**repeat**	**while**
قد تمّمت التكرارية على الأقل	0 مرّة	1 مرة واحدة	0 مرّة
اختبار التوقف		بعد التكرارية	قبل التكرارية

مثال : إحصائيات

في جملة دراسات إحصائية ، يوجد حاجة لمعرفة عدد الفراغات (تباعد) في نص مكوّن على الشكل التالي :

1 ـ يوجد ثلاث فقرات ، مؤلفة من جُمَل ، تنتهي الفقرة بجملة لا تحتوي على تباعد ومؤلفة جزءاً من الفقرة .

2 ـ تتكون كل جملة من عدة سمات بما فيها التباعد وتنتهي بالسمة . التي لا تدخل ضمن نطاق الجملة .

نودّ إذن الحصول بالنسبة لكل فقرة على متسلسلة اعداد التباعدات والسمات في كل جملة ، مفصولة بخط وَصْل (-)
مثلاً في حال المعطيّـة التالية :

EN FAIT. IL Y A T IL. UN. RADEAU. AVEC UN S. DEDANS

فإن النتيجة تكون : 1-7, 4-11, 0-2, 0-6, 2-9, 0-6

ملاحظة : ينوّط النوع سمة في لغة الباسكال «char» ؛ يمكن قراءة وكتابة سمات كما يمكن مقارنتها في ما بينها .

تعريفات		معجم
نتيجة = عالج لِـ فقرة من 1 إلى 3	1	ـ عالج (نص) لـ فقرة ـ فقرة (صحيح)
		لِـ فقرة ← عالج
عالج = أكتب nb Car, '—', nb Espaces لِـ جملة من 1 متضمّناً nb Espaces = 0	1	ـ nb Espaces (صحيح) لـ جملة : عدد التباعد في جملة ـ nb Car (صحيح) لِـ جملة : عدد السمات في جملة ـ جملة (صحيح)
		لِـ جملة ← nb Car, nb Espaces
nb Espaces = آخر nb Espaces = $\overline{x + \text{nb Espaces}}$ لِـ c من 1 طالما '.' <> carac nb Car = آخر nb Car = $\overline{+\text{ nb Car}}$ لِـ c من 1 طالما '.' <> carac أول nb Espaces = 0 أول nb car = 0	3 3 1 1	ـ x (صحيح) لِـ c:1 إذا كانت هي تباعد ـ carac (سمة) لِـ c: سمة ذات الرقم c ـ c (صحيح)
		لِـ c ← x, car ac
x = 1 إذا ' ' = carac، 0 وإلاّ carac وفقاً لمعطيّـة	2 1	

إن المسألة التي فيها نستبدل سمة بعدد صحيح

نقطة بـ 1 −

تباعد بـ 0

تبدو أسهل . لكن لا شيء يفيد ، فتعقيد التحليل يبقى كما هو (مع الأخذ بعين الاعتبار أن معالجة القيم الرقمية يمكن أن تظهر مألوفة أكثر) .

البرنامج

```
program statistiques(input,output);
{ عد التباعدات في نص مؤلف من ثلاث فقرات ، مؤلفة من جمل تنتهي الفقرة بجملة
دون تباعد. تنتهي الجملة بنقطة ، التي لا تدخل ضمن الجملة . يطلب عد نسب التباعدات ، جملة بجملة }
var paragraphe,phrase,c:integer;     { دلائل }
    nbEspaces,nbCar:integer;  { عدد التباعدات ، السمات في كل جملة }
    x:integer;    { لكل سمة ، 1 إذا كان تباعداً }

    carac:char;   { لكل سمة ، قيمتها }
begin
  for paragraphe:=1 to 3 do
PARA( فقرة                   repeat
                               nbCar:=0; nbEspaces:=0;
          أولي  ───────►       read(carac);
                               if carac=' ' then x:=1 else x:=0;
          جملة                 while carac<>'.' do begin
إنتاج ───► إستعمال ──────►        nbCar:=nbCar+1; nbEspaces:
                                      =nbEspaces+x;
          إنتاج ───────►       read(carac);
          (إقصاء)              if carac=' ' then x:=1 else x:=0
                             end;
إستعمال ─────────────────►   writeln(nbEspaces,' - ',nbCar)
( تضمين )                  until nbEspaces=0
                     end.
```

(Statistiques : إحصائيات ؛ compter = عدّ ؛ espaces : تباعد ؛ texte : نص ؛ paragraphes : فقرة ؛ phrase : جملة ؛ point : نقطة) .

6.1 ـ جداول (Tables)

في المثال السابق ، كان بالإمكان دمج التكراريتان (المعرّفتان لـ nb Espaces وNb Car) ؛ كل معطيّـة مقروءة تم في الحال إستعمالها في معالجتين مستقلتين . حالات أخرى يمكن كذلك أن تظهر :

- لوضع فاتورة لعدة سلع مختلفة ، فسنراجع التعرفة ، الممثّلة بواسطة **جدول** : دالة ذات قيم حقيقية (الأسعار) لمتغير صحيح (إسنادات السلع) ؛
- لكي نقدّر تشتّت القيم في متسلسلة ، نحسب الإنحراف المعياري (écart type) ، هذا ما يستتبع إستعمال المتسلسلة مرتين ؛ لذا نحفظها في **جدول** : تركيب أسس الحفظ ومن ثم إستعماله لمرات عدة .
- لكي نحسب التكرُّر الخاص بسمات في نص ، نؤلف متسلسلة جداول لقياس المسار في النص : تركيب أسس الحفظ بواسطة التكرارية ، من ثم إستعماله لمرة واحدة فقط ؛
- لكي نبرمج عمليات خاصة بالمصفوفات ، نعرِّف جداولاً ذات عدة أبعاد ؛

ـ لكي نبرمج جدول تقرير ؛ أي صيغة شرطيّـة ذات عدة خيارات ، فإنه يمكن إستعمال ، تبعاً للحالة ، جدولاً للصيغ الشرطية ، أو كذلك عبارة خيارات .
في كل هذه الأمثلة ، إن الإنطلاق من النتيجة يسهّـل التحليل .

جدول : دالّـة لمتغيّـر صحيح

لكي نضع فاتورة ، فإننا نجمع أثمان كل سلعة مطلوبة :

$$total = \sum_{p} prixUnitaire_p \times quantité_p$$

(total : مجموع ؛ Prix unitaire : سعر إفرادي ؛ quantité : كمّية) .

حيث أن الـ P هي اسنادات السلع .

في الحالة التي يكون فيها الإسناد عدد صحيح ، موجود بين 1 و10 مثلاً ، فإن سلسلة الـ 10 أسعار إفرادية توصف في المعجم بـ « سعر إفرادي [1... 10] (حقيقي) » ويكتب السعر الإفرادي الخاص بالعنصر P « سعر إفرادي [P] » .

بالفعل فإن المراد من سلسلة الـ 10 قيم حقيقية سعر إفرادي [1] ، سعر إفرادي [2] ، . . . سعر إفرادي [10] هو إمكانية الكتابة السهلة (معالجة الـ 10 معرّفين سعر إفرادي 1 ، سعر إفرادي 2 ، إلخ . . . هو أمر صعب ومتعب) .

إن التنويط المقابل لذلك في لغة الباسكال هو :

prixUnitaire : **array** *[1..10]* **of** *real;*
et *prixUnitaire [p]*

إذا تُـفق على أن المشتريات تنتهي بالمشترىَ « تنكة » للسلعة 0 ، فإننا نحصل على :

تعريفات		معجم
نتيجة = أكتب مجموع مجموع = آخر مجموع = مجموع + ثمن لِـa من 1 طالما سلعة <> 0 أول مجموع = 0.0		ـ مجموع (حقيقي) : الثمن المتوجب دفعه ـ الثمن (حقيقي) لـ a : الثمن الإجمالي للـ aième مُشْتَرَى ـ a (صحيح) ـ سلعة (صحيح) لِـ a : اسناد الـ aième مشترى
		لِـ a ← ثمن ، سلعة
ثمن = كمية * سعر افرادي [سلعة]		ـ كمّية (صحيح): عدد الأشياء المشتراة

إذا كان مكان الـ « سعر افرادي [سلعة] » هو فعلاً في الجدول الثانوي ، فإن مكان سلسلة الأسعار الإفرادية « سعر افرادي [1 ...10] » هو في الجدول الأساسي : إن « سلعة » هي التي تتعلق بالتكرارية وليس « سعر إفرادي » . إذن نضيف على معجم الجدول الأساسي : سعر إفرادي [1 .... 10] (حقيقي) : تعرفة .

ونهي الجدول الثانوي بـ :

كمية ، سلعة وفقاً لمعطيّة		

أخيراً يبقى أن نعرّف السعر الإفرادي في الجدول الأساسي :

سعر إفرادي = سعر إفرادي [i] وفقاً لمعطيّة لـ i من 1 إلى 10		ـ i (صحيح)

البرنامج

```
program facture(input,output);
{ فوترة مُشترى عدة سلع ، مع إستعمال تعرفة }
var total:real;          { الثمن المتوجب دفعه }
    prix:real;           { لكل مشترى ، ثمنه }
    a,i:integer;
    produit,quantite:integer;  { لكل مشترى ، إسناده في التعرفة ، وعدده }
    prixUnitaire:array[1..100] of real;   {تعرفة }
begin
  for i:=1 to 10 do read(prixUnitaire[i]);
  total:=0.0;
  read(quantite,produit);
  while produit<>0 do begin
    prix:=quantite*prixUnitaire[produit];
    total:=total+prix;
    read(quantite,produit)
  end;
  writeln(total)
end.
```

(achat : مشترى ؛ produit : سلعة ؛ tarif : تعرفة ؛ quantité : كمية ؛ total : مجموع)

إذا كان لدينا المعطيات :

5 2 3 8 1 1 2 8 9 4 1 2 2 4 1 9 5 5 0 0

تكون النتيجة : 32

جدول : إستعمال

على متسلسلة S من N قيمة ، نعرِّف :

الوسط الحسابي : $m = \frac{\sum_{i=1}^{N} S_i}{N}$

الإنحراف المعياري ، تقدير إنحراف القيم من وسطهم الحسابي

$$e = \sqrt{\frac{\sum_{i=1}^{N}(S_i - m)^2}{N}}$$

سنستعمل إذن الوسط الحسابي m لحساب الإنحراف المعياري : S ستكون جدولاً .

تعريفات		معجم
نتيجة ≐ أكتب e, m	9	ـ m (حقيقي) : وسط حسابي
m = S1 / N	5	ـ e (حقيقي) : إنحراف معياري
e = Sqrt (s2/N)	8	ـ S1 (حقيقي) : مجموع القيم
S1 = dernier S1 = $\overline{S1}$ + S [i] pour i de 1 à N	4	ـ N (صحيح) : عدد القيم
	3	
أوّل S1 = 0.0	1	ـ S2 (حقيقي) : $\sum (Si - m)^2$
وفقاً لمعطيّة N		ـ S [1...N] (حقيقي) : جدول الـ N قيمة
S2 = dernier S2 = $\overline{S2}$ + (S [j] − m) * (S [j] − m) pour j de 1 à N	7	ـ i (صحيح)
أوّل S2 = 0.0	6	ـ j (صحيح)
S = S [K] وفقاً لمعطيّة لِـ K من 1 إلى N	2	ـ K (صحيح)

البرنامج

```
program ecartType(input,output);
{ حساب الوسط الحسابي والإنحراف المعياري }
var m:real;          { وسط حسابي }
    e:real;          { انحراف معياري }
    S1:real;         { مجموع القيم }
    S2:real;         { مجموع مربّعات الانحرافات }
    N:integer;       { عدد القيم }
    S:array [1..1000] of real; { جدول الـN
                     قيمة مُكبّر أبعاده }
    i,j,k:integer;
```

```
                begin
تركيب  →       read(N); for k:=1 to N do read(S[k]);
استعمال →        S1:=0.0; for i:=1 to N do S1:=S1+S[i]; m:=S1/N;
إستعمال →        S2:=0.0; for j:=1 to N do S2:=S2+(S[j]-m)*(S[j]-m);
                  e:=sqrt(S2/N);
                  writeln(m,e)
                end.
```

(انحراف معياري = écart type)

في لغة الباسكال ، تكون الحدود المصرَّحة لدليل الجدول من الثوابت ؛ من هنا تكبير أبعاد الجدول S .

جدول : تكوين

في نص مؤلف من السمات '0', '1', ... '9' فقط ومنتهي بنقطة ، نسعى لحساب التكرر الخاص بالسمات ـ الأرقام .

ملاحظة : 1 . يوجد على النوع سمة (char) علاقة ترتيب relation d'ordre :

$$'0' < '1' < \dots < '9'$$

ودالّتين succ وpred تسمحان بالانتقال من سمة إلى أخرى أكبر مباشرة (أو أصغر) :

$$succ\ ('0') = '1' \qquad pred\ ('5') = '4'$$

2 ـ إن النوع سمة ، كما النوع صحيح ، هو نوع ترتيبي ، يُمكِن :

- تدليل الجدول بعدد صحيح ، أو بسمة .
- كتابة تكرارية تعمل على فترة من الأعداد الصحيحة أو من السمات (أنظر الفصل 2) .

المطلوب هنا تكوين متسلسلة من الجداول لقياس مسار النص :

معجم		تعريفات
ـ c (سمة)	6	نتيجة = أكتب c ، F [c] لـ c من '0' إلى '9'
ـ F ['0'... '9'] (حقيقي) : تردّد	5	***F = F [x] = N [x]/qté* pour *x* de *'0'* à *'9'***
ـ N ['0' ... '9'] (صحيح) : عدّ ـ x (سمة) ـ qté (صحيح) : عدد السمات ـ i (سمة) لـ carac : سمة متداولة	2	***N* = dernier *N ['0'..pred(i)]* = $\overline{N}$ *['0'..pred(i)]* =** ***N [i]* = $\overline{N}$ *[i]* + *1*** ***N[succ(i)..'9']* = $\overline{N}$ *[succ(i)..'9']*** **pour *carac* de *1* tant que *i* <> *'.'***
ـ carac (صحيح) : دليل السمة المتداولة	1	**premier *N* = *N [j]* = *0* pour *j* de *'0'* à *'9'***
ـ j (سمة)	4	***qté* = dernier *qté* = $\overline{qté}$ + *N [k]* pour *k* de *'0'* à *'9'***
ـ k (سمة)	3	أوّل qté = 0

لـ carac ← i

| 1 |

i وفقاً لمعطيّة

هذه المسألة تؤدي إلى مظاهرة عمليات على جداول كاملة (تعيين ، قسمة بقيمة في القاعدة 5 الجمع في القاعدة 2) ؛ إن استعمال العمليات على جداول كاملة هي أداة التحليل العادي ؛ هنا ، نصفهم لكي نقترب من التنويطات المستعملة في لغة الباسكال .

البرنامج

```
program frequences(input,output);
{ حساب تردد سمات ( أرقام) في نصّ منتهي بنقطة }

var c,j,k,x:char;
    F:array['0'..'9'] of real;          { تردّد }
    N:array['0'..'9'] of integer;       { عدّ }
    qte:integer;              { عدد السمات }
    i:char;                   { سمة متداولة }
    carac:integer;            { ترتيب السمة المتداولة }
begin
  for j:='0' to '9' do N[j]:=0;
  read(i);
  while i<>'.' do begin
    N[i]:=n[i]+1;
    read(i)
```

```
   end;
   qte:=0; for k:='0' to '9' do qte:=qte+n[k];
   for x:='0' to '9' do F[x]:=N[x]/qte;
   for c:='0' to '9' do writeln(c,F[c])
end.
```

ملاحظة : إن التكريرات المستعملة على x وc يمكن لها أن تندمج وتصبح :

for *x:* = *'0'* **to** *'9'* **do** *writeln (x, N [x]/qté)*

إذا كانت المعطيات هي :

3 1 3 7 9 1 3 4 1 1.

فإن النتيجة تصبح

1	0.4
2	0.0
3	0.3
4	0.1

إلخ

إن المسألة المحصول عليها باستبدال سمة بـ رقم
نقطة بـ عدد ما
يمكن أن تظهر أكثر سهولة (يجب التجريب) ، لكنها فعلياً بنفس مستوى التعقيد .

مصفوف (Matrice)

نسمي بالمصفوف ، الجدول ذي البعدين ، أي المدلّل بدليلين . مثلاً جدول العلامات على 20 لـ x في y مادة هو مصفوف :

تلميذ \ مادة	1	2	j	y
1	علامة 1,1	علامة 1,2	.....	علامة 1,y
2	علامة 2,1	علامة 2,2	.....	علامة 2,y
i	⋮	⋮	علامة i,j	⋮
x	علامة x,1	علامة x,2	.....	علامة x,y

تنويطات في التحليل

note [i, j]　　　note [1... x, 1... y] (حقيقي)

وفي لغة الباسكال

note [i, j] **array** [1...x, 1... y] **of** real

(note = علامة)

حيث أن x وy هما ثوابت . فيما يلي سنفترض وجود تلميذين وثلاثة مواد . لكي نستثمر هذا المصفوف ، يوجد عدة عمليات ممكنة :

- إن حساب الوسط الحسابي للقيم على السطر i ، يعني حساب الوسط الحسابي للتلميذ i :

$$\textbf{moyenneElève}_i = \left(\sum_{j=1}^{y} \textbf{note}_{i,j}\right)/y$$

(moyenne = وسط حسابي ؛ éléve = تلميذ)

تعريفات		معجم
نتيجة = أكتب Si / y لِـ i من 1 إلى x	4	- Si (حقيقي) لِـ i : مجموع التلميذ i
y = 3	2	- y (صحيح) : (3) عدد المواد
x = 2	1	- i (صحيح) - x (صحيح) : (2) عدد التلاميذ
علامة وفقاً لمعطيّـة	3	- note [1...x, 1...y] (حقيقي)

لِـ i ← S

غير متغير

صاعد

Si = آخر Si = Si + علامة [i, j] لِـ j من 1 إلى y	2	
أول Si = 0.0	1	- j (صحيح)

يجب على عملية قراءة المصفوف أن تكون مشروحة كاملاً في لغة الباسكال .

x وy تلعبان دور الثوابت ؛ سنستعملهما كما هما في البرنامج وذلك بالتصريح عنهما

const y = 3 ; x = 2

قبل تصريح المتغيرات .

البرنامج

```
program moyenneEleves(input,output);
{ معدّلات x تلميذ في امتحان من y مادة }
const y=3; { مواد }                x=2; { تلامذة }
var Si:real;          { لكل طالب ، مجموع }
    i,j,il,jl:integer;
    note:array[1..x,1..y] of real;
begin
  for il:=1 to x do
    for jl:=1 to y do
      read (note[il,jl]);
  for i:=1 to x do begin
    Si:=0.0; for j:=1 to y do Si:=Si+note[i,j];
    writeln(Si/y)
  end
end.
```

إذا كانت المعطيات هي :

11 16 6 9 12 6

فالنتيجة تصبح : 9 11

ـ إن حساب الوسط الحسابي للقيم على العامود j ، يعني حساب الوسط الحسابي المحصّل عليه في مادة من قبل مجموع التلاميذ :

$$\text{moyenne Matière}_j = \left(\sum_{i=1}^{x} \text{note}_{i,j}\right)/x$$

(مادة = Matière)

نتيجة = أكتب Sj / x لِـ j من 1 إلى y

Sj = آخر Sj = Sj + علامة [i, j] لِـ i من 1 إلى x

```
...
for j:=1 to y do begin
  Sj:=0.0; for i:=1 to x do Sj:=Sj+note[i,j];
  writeln(Sj/x)
end
...
```

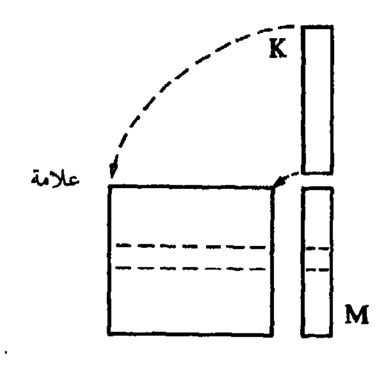
مع المعطيات

11 16 6 9 12 6

النتيجة 6 14 10

ـ إن القيام بضرب المصفوف علامة بمتّجه (Vecteur) (جدول ذي بعد واحد) ذي y

قيمة ، يعني حساب الوسائط الحسابية للتلاميذ آخذين بعين الاعتبار معاملات المواد :



$$M_i = \sum_{j=1}^{y} note_{i,j} \star K_j$$

$$(\sum_{j=1}^{y} K_j = 1 \text{ مع})$$

...
— *K [1..y]* (حقيقي)
...

نتيجة = أكتب M لِـ i من 1 إلى x
وفقاً لمعطية K

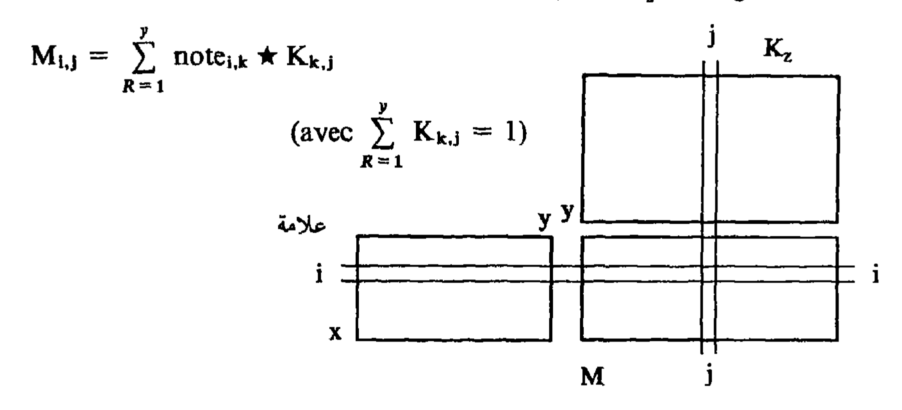
. . . .
M = آخر M = $\overline{M}$ + علامة [i, j] * K [j]
لَـ j من 1 إلى y
. . . .

```
...
  for i:=1 to x do begin
    M:=0.0; for j:=1 to y do M:=M+note[i,j]*V[j];
...
```

- إن القيام بضرب المصفوف علامة بمصفوف ذي y سطر وz عمود ، يعني حساب الوسائط الحسابية لكل تلميذ في z تلاعب بالمعاملات :

$$M_{i,j} = \sum_{R=1}^{y} note_{i,k} \star K_{k,j}$$

$$(\text{avec } \sum_{R=1}^{y} K_{k,j} = 1)$$



```
for i:=1 to x do
  for j:=1 to z do begin
    M[i,j]=0.0;
    for k:=1 to y do
      M[i,j]:=M[i,j]+note[i,k]*V[k,j]
  end
```

خيارات

يمكن أن يكتب التعريف الشرطي بشكل جدول تقرير ذي خيارين :

$x = y$ **si** *condition*, z **sinon**

شرط →	متوفر	عاجز
$x =$ ←	y	z

يمكن بسهولة تخيّل قرار بثلاثة خيارات :

a مقارنة مع b	$a < b$	$a = b$	$a > b$
$x =$	$e1$	$e2$	$e3$

أو بـ n خيار :

إذا سمة c →	+	−	★	/	...
إذن r = ←	$a + b$	$a - b$	$a \star b$	a/b	

تكتب التعريفات المقابلة كما يلي :

$$x = e1 \text{ si } a < b,\ e2 \text{ si } a = b,\ e3 \text{ si } a > b$$

$$r = a+b \text{ si } c = '+',\ a-b \text{ si } c = '-',\ a \star b \text{ si } c = '\star',\ a/b \text{ si } c = '/'$$

حيث أن سرد الشروط المتتالية يجب أن يشمل كل الإمكانيات ، لأنه لا يوجد قرار « وإلا »

في لغة الباسكال ، تكون الترجمة دائماً ممكنة بواسطة إختبارات شلشلية

```
if a <b then x:=e1
else if a=b then x:=e2
     else if a>b then x:=e3
          else ...
```

عندما تدور مختلف الشروط حول تعداد قيم تعبير من النوع المرتّب (type ordinal) (صحيح ، بولي ، سمة) ، فإننا نكتب :

case expression **of**
  valeur$_1$: énoncé$_1$;
  valeur$_2$: énoncé$_2$;
   ...
end

(عبارة = énoncé ؛ قيمة = valeur ؛ تعبير = expression)

لنفرض مثلاً أننا نريد كتابة برنامج محساب صغير متمّم للأربع عمليات على أعداد حقيقية .

تعريفات			معجم
نتيجة = أكتب r		3	ـ r (حقيقي) : النتيجة المحسوبة
	r = a + b Si c = '+',	2	ـ a (حقيقي) : متأثر 1
	a − b Si c = '—' ,		ـ b (حقيقي) : متأثر 2
	a * b Si c = ' * ',		
	a / b Si c = ' / '	1	
وفقاً لمعطيّة c, b, a			c (سمة) : رمز حسابي

البرنامج

```
program calculette(input,output);
{ الأربع عمليات على اعداد حقيقية }
var r,a,b:real;   { النتيجة ، المتأثرين }
    c:char;       { المؤثر : + - * أو / }
begin
  read(a,c,b);
  case c of
    '+': r:=a+b;
    '-': r:=a-b;
    '*': r:=a*b;
    '/': r:=a/b;
  end;
  writeln(r)
end.
```

(محساب صغير : calculette)

مع المعطيات : 20 * 317.961 النتيجة : 635.922

المعطيات : 1.0 / 0.0 النتيجة : مَأْلَكَةُ الغلط (message d'erreur) ، متوقفة على الحاسب المستعمل

7.1 ـ إختيار طريقة

تسمح الطريقة الإستنتاجية بانطلاقة جيدة في البرمجة وكذلـك هي وسيلة نظريـة خصبة .

بالنسبة للمبرمج ، تكمن فائدتها في جملة ما ، في أنها تستوجب تفكيراً « على ما نريد صنعه » ، تفكيراً يؤدي الى استنتاجات على « كيف يجب صنعه » . هذا المدخل يُوصِل عامة إلى برامج صحيحة إبتداء من التجربة الأولى (هذا ما هو نادراً إذا كنا نبرمج دون إتباع طريقة ما) .

إن التنفيذ اليدوي بالنسبة لبعض أنواع البرامج ، يصبح سريعاً شاقاً للغاية ؛ يجب إذن أن نحفظ المهم من أفكار الطريقة وأن نترك جانباً التحليل التقني . إن الطريقة المقتَرَحَة لتكوين برنامجاً تبقى بالفعل فعّـالة بالكامل إذا تم إهمـال التفاصيـل الغير متـوافقة مـع المسألة .

إذا عملنا بهذا الشكل فإننا سنقترب من طرق تحليل أخرى . الأكثر شمولية والأكثر شهرة هي فيما يظهر البرمجة بطريقة التدقيق المتتالي (raffinements successifs) التي سوف نتناولها تباعاً في سياق النص .

8.1 ـ تمارين

1 ـ أوجد الثمن المتوجب دفعه عند تسليم كمية من الفيول . يجب الأخذ بعين الاعتبار قيمة الضريبة المتوجبة وحسماً بقيمة 3% إذا تعدى المبلغ الإجمالي 2500 فرنك .

2 ـ حِلّ المعادلة من الدرجة الأولى $ax + b = 0$ مع الأخذ بعين الإعتبار الحالة التي تكون فيها a مساوية لصفر .

3 ـ حلّ النظام الخطّي للمعادلتين ذات المجهولينْ :

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

وذلك بتطبيق صِيَغ كرامر (Cramer) :

$$x = (ce - bf)/(ae - bd), \quad y = (af - cd)/(ae - bd)$$

4 ـ إحسب المال الواجب إرجاعه بواسطة موزّع اوتوماتي محتوٍ على عدد غير محدد من القطع النقدية : 10 فرنك ، 5 فرنك ، 2 فرنك و1 فرنك ، يجب أن نقلّل من عدد القطع الواجب إرجاعها .

5 - إطبع جدولاً للـ N قوة متتالية للقيمة x .

6 - إطبع جدولاً لقيم المتسلسلة U حيث :

$$u_i = (i + 1)/(i \star i + i + 1) \text{ pour } 0 \leqslant i \leqslant n.$$

7 - احسب $S2_i = \sum_{n=1} n^2$

8 - احسب عامليُّ القيمة الصحيحة المعطيّة (factorial)

$(F_n = (F_{n-1}) \star N)$

9 - إحسب في $\mathbb{R}^n$ المسافة بين نقطتين A و B محدّدتين باحداثياتهن

$$A_1, A_2, \ldots A_n \text{ et } B_1, B_2, \ldots B_n.$$

$$d(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_j)^2}$$

10 - إقرأ نصاً ينتهي بنقطة ، وله على الأكثر 40 سمة ومن ثم اكتبه بالمقلوب على أن ينتهي النص المكوّن حينئذٍ بنقطة . فمثلاً «.lire un texte» كمعطيّة تصبح بالمقلوب «etxet nu eril.» .

11 - اضبط (Normaliser) متسلسلة v من القيم الحقيقية : المراد هو إعادة القيم على الفترة [0..1] بواسطة تطبيق خطّي (application linéaire)

$$u_i = kv_i$$

12 - تسمح صيغة جون ماشين (John Machin) بحساب بي (π)

$$\pi = 4 \sum_{0}^{\infty} \frac{(-1)^n}{2n + 1} \left(\frac{4}{5^{2n+1}} - \frac{1}{239^{2n+1}} \right)$$

إنها بالفعل عملية تطوير π المساوية لـ :

(π = 16 قوس ظل $\frac{1}{5}$ - 4 قوس ظل $\frac{1}{239}$)

الفصل الثاني

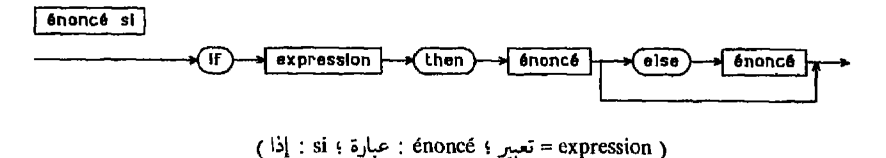
قواعد اللغة

0.2 ـ كتابة برنامج

عندما نتكلم عن كتابة برنامج ، فإننا نعني حينها (على الأقل) شكلين : النحـو والمعنى .

المعنى (Sémantique) ويقصد به كل ما له علاقة بالمدلول ، إنه ما يريد البرنامج قوله ، إنه تسلسل الحسابات التي تؤدي إلى النتيجة . لا تصبح النتيجة سليمة إلّا إذا كان تحليل المسألة (التي يَحُلّـها البرنامج) قد تم بشكل صحيح ، إمّا بطريقة صورِيَّة ، مثل الطريقة الإستنتاجية المشروحة سابقاً ، إمّا بالمهارة التي هي ثمرة التجربة . لا تنطبق هذه الحالة الأخيرة إلّا على المسائل السهلة التي يكون قد حُلَّ مثلها سابقاً .

النحو (Syntaxe) ويقصد به كل ما له علاقة بوظيفة وتنظيم الرموز ، إنه الطريقة التي يجب أن يُكْتَب بها البرنامج لكي يصبح مفهوماً من قبل الآلة . يتم تحديد نحو البرنامج بواسطة قواعد اللغة المرسومة هنا بواسطة مخطّطات :



(expression = تعبير ؛ énoncé : عبارة ؛ si : إذا)

فإذن « if a = b then c : = a / 2 » هي من الناحية النحوية سليمة ، بينما « if a: = d − e else x − y > 0 » هي ليست كذلك وسيتم الإعلان عنها من قبل الآلة كجملة غير مفهومة .

نُنوِّط في المخطَّط داخل مستطيل ما هو معرَّف في مكان آخر ، ونُحيطُ بدائرة الرموز الأساسية للّغة (الكلمات الدليليّـة «Key-word» والرموز الخاصة مثل : = أو ؛) .

معلومات سابقة

يتناول هذا الفصل ما هو ضروري ولا غنى عنه لكتابة برنامج في الباسكال . إنه لا يستلزم معلومات أخرى عن التحليل غير التي تناولها الفصل الأول أو على الأقل ممارسة للبرمجة في لغة أخرى كالـ BASIC أو الـ FORTRAN مثلاً .

سننطلق من التركيبة العامة للبرنامج ، وسنوضح تباعاً العناصر ، دون أن نسعى للإحاطة بكل اللغة (هذا ما سيتم في الفصل التالي) ولا إلى التفكير بالتحليل أو الخوارزمات أو مجمّع المعطيّات .

إنه فصل يعتمد على الوصف ، حيث تتطابق فيه الأمثلة مع تشكيلات أو مع تماثلات بارزة أكثر مما هي تنطبق على برامج كاملة .

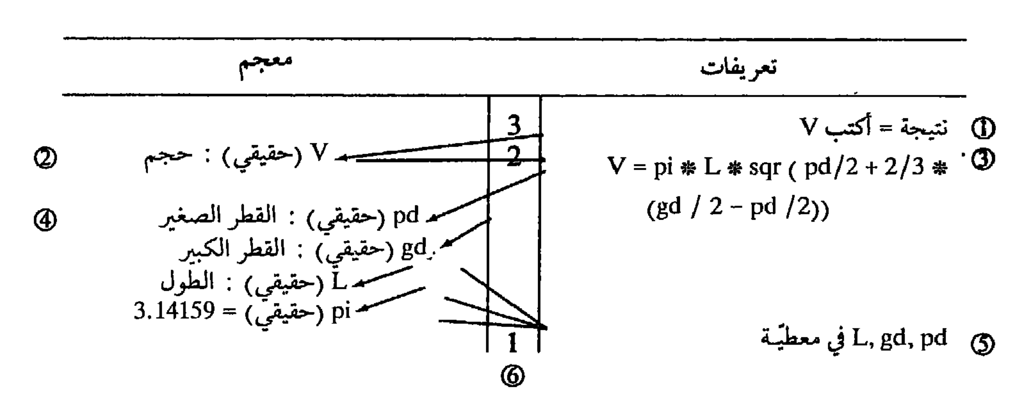
1.2 ـ مفاهيم مبدئية

مثال : برميل

إن المسألة « إحسب حجم برميل » تؤدي الى المواصفة « إحسب حجم البرميل ذي الأبعاد التالية : الطول L ، القطر الكبير D ، القطر الصغير d على أن يحسب حجمه تبعاً للصفة التالية :

$$V = \pi L \left[\frac{d}{2} + \frac{2}{3}\left(\frac{D}{2} - \frac{d}{2}\right)\right]^2 \text{»}$$

لتكوين التحليل ، ننطلق من تعريف النتيجة (أكتب V) ، لحساب V ، نستعمل L ، d وD الذين هم معطيات :



في هذا التحليل قمنا بتخطيَة (Linéarisation) الصيغة الرياضية والتحديد بأن pi هي ثابتة : إننا نستبق على نحو الباسكال . كذلك فإننا أشرنا بـ pd إلى d وgd إلى D ذلك لأن d وD سيكونان كتابتان مختلفتان لنفس المعرّف .

إن النقل إلى لغة الباسكال يؤدي إلى تمييز **التصريحات** ، المتوافقة مع المعجم ، التي تحدد خواصّ **المعرّفين** (pi ثابتة ، D, d وL معرّفين **لمتغيرات** ذات قيم في مجموعة الأعداد الحقيقية) **والعبارات** ، المتوافقة مع التعريفات ، التي تحدّد الحسابات الواجب إجراءها على المتغيرات والثوابت :

البرنامج

```
program tonneau(input,output);
{ حساب حجم البرميل }
const pi=3.14159;
var V:real;        { حجم }
     pd:real;      { القطر الصغير }
     gd:real;      { القطر الكبير }
     L:real;       { الطول }
begin
  read(pd,gd,L);
  V:=pi*L*sqr(pd/2.0+2.0/3.0*(gd/2.0-pd/2.0));
  writeln (V)
end.
```

(tonneau : برميل)

تصريحات وعبارات (Déclarations et énoncés)

يتألف البرنامج من **تصريحات تحدِّد الأدوات** المعالَجة من قبل العبارات ، وذلك بتحديد طرق استعمالها (ثابت ، متغير ، . . .) نوعيتها (عدد صحيح ، عدد حقيقي ، . . .) والمعرِّف المعبِّر عنها (..., V, pi) ؛

وعبارات تحدِّد الأفعال المبدئية الواجب إجراءها (قراءة ، مقارنة ، كتابة ، تعيين . .) وسَلسَلتها : متتالية ، شرطيّة ، . . .

ملاحظة : يطلق إسم « تعليمة » على العبارة . إننا نتبع النظم AFNOR التي تستعمل « العبارة ».بشكل عام ، يجب التصريح عن كل معرِّف قبل استعماله .

بعض المصطلحات (un peu de Vocabulaire)

الثابت هو قيمة غير قابلة للتغيير من قبل البرنامج . يجب تمييز الثوابت الصحيحة (4 ، 3- ، 0) ، الثوابت الحقيقية (3.14159) ، الثوابت المسلسلة ('volume = ' هو سلسال من سبع سمات) . يمكن تمثيل الثابت بمعرّف لثابت (pi) ، محدَّد بواسطة تصريح لثابت (مُدخل بالكلمة الدليل **const**) .

في مقابل الثابت ، فإن قيمة **المتغير** قابلة للتغيير من قبل البرنامج .

المعرِّف هو الإسم المطلق على أداة ؛ يوجد معرّفين لثابت (pi)، لمتغير (v,pd, gd)

لإجراء (writeln, read) . . . يتم حجز بعض المعرّفين لغايات خاصة ، إنها الكلمات الدليليّـة (.., Const, program) .

إن مجموعة القيم التي يمكن أن تأخذها أداة هي نوعها . يمكن للنوع أن يحدد من قبل البرنامج ، أو أن يكن محدّد سلفاً (integer, real : حقيقي ، صحيح) .

إن عبارة التعيين (. . . * l * V: = pi) تعيد تعريف القيمة الملحقة بـالمتغيِّـر المعيَّـن (V) نتيجة التقييم لتعبير (. . . * l * pi) .

التمثيل الحَرفي (représentation typographique)

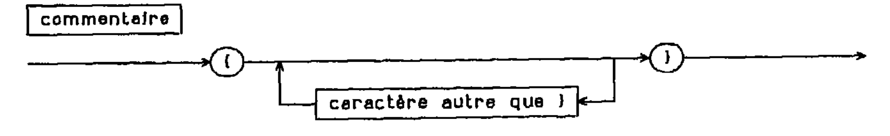
إن كل سمة ظاهرة خارج إطار ثابتة مسلسلة (' = volume' أو 'H') لا تجد معناها قد تغيّـر بفعل تغيير في التمثيل الحرفي . بوجه خاص ، يمكن بالنسبة لمعرف تمثيل حرفٍ بالشكل الصغير (minuscule) أو بالشكل الكبير (majuscule) : فمثلاً ,longueur donde longueur dOnde وLONGUEURdONDE هم ثلاثة كتابات لنفس المعرّف ؛ كذلك 14 ، 14 ، **14** و14 هم نفس الثابت .

تبعاً للأدوات المستعملة ، يمكن توفير طريقة أو عدة طرق تلاعب بالسمات الحرفية (غليظ صغير ، كبير، . . .) ؛ عامة فإن الأحرف الكبيرة هي متوفرة دوماً .

كل تلك الطرق تهدف الى تأمين قراءة جيدة للبرنامج وهذا ما نستعمله في كتابنا هذا عندما نكتب البرامج إلاّ في حالة البرامج المختبرة الكاملة حيث أن الطابعة المستعملة لا تسمح بتلاعب في التمثيلات الحَرفيّـة .

ملاحظات (Commentaires)

تستعمل الملاحظة بهدف تسهيل فهم البرنامج من قبل القارىء العادي ، بينما تبقى دون أي تأثير فيما خصّ مدلول البرنامج بالنسبة للحاسب الآلي .



(commentaire : ملاحظة ؛ caractère autre que : سمة غير)

إن الملاحظات ، في الحالة الراهنة للتكنولوجيا ، هنَّ غير مفهومات من قبل الحاسب الآلي (هذا ما يدعو إلى إهمالهن) ، بينما هن ربما الجزء الأساسي في البرنامج .

يمكن أن يتم إدخال الملاحظات في أي مكان حيث يمكن ، أو يجب ، وضع تباعد (سمة فراغ ' ') .

تباعدات (Espaces)

إن الملاحظات ، التباعدات (مـا عدا تلك التي تـوجد في سلسـلات السمات) ونهايات السطور هن فواصل لوحدات من مفردات اللغة . يمكن أن تظهر تلك الفواصل :

- مرة واحدة على الأقل بين وحدتين متتاليتين واللتين تكونان معرّفين ، كلمات دليلية ، وسومات أو أعداد دون علامة ؛
- مرة واحدة على الأكثر بين وحدتين من مفردات اللغة .

يمنع وضعهم داخل وحدات مفردات اللغة ؛ فمثلاً لا يمكن إدخال فراغ ، ملاحظة أو نهاية سطر في معرف ما ، يجب أن نكتب « : = » دون أي تباعد .

غالباً ما نوجز هذه القواعد بالقول « ان » لغة الباسكال هي لغة ذات نسق حر » .

ملاحظة : نجد في الملحق 3 قائمة الرموز الخاصة والكلمات الدليليّـة .

2.2 ـ التكوين الإجمالي للبرنامج

البرنامج (Programme)

تظهر مختلف أجزاء البرنامج ضمن ترتيب محدّد مفروض من قبل قواعد النحو :

1 ـ عنوان البرنامج

2 أقسام التصريحات .

3 ـ قسم التعابير .

يطلق إسم جسم على قسم التعابير ، وفِدْرَة على أقسام التصريحات والتعابير .

تتألف أقسام التصريحات من :

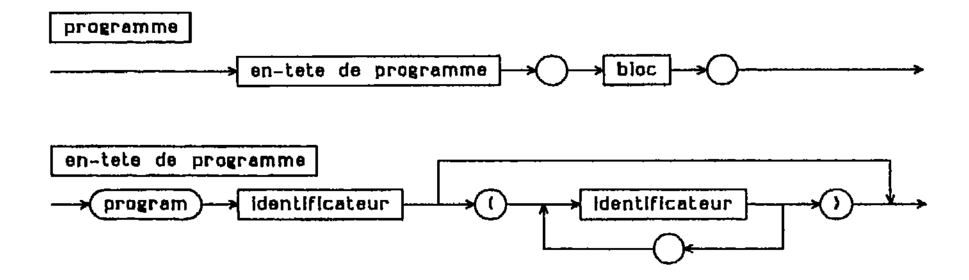
1 ـ قسم تصريح الوسومات (أنظر 3.3.3)

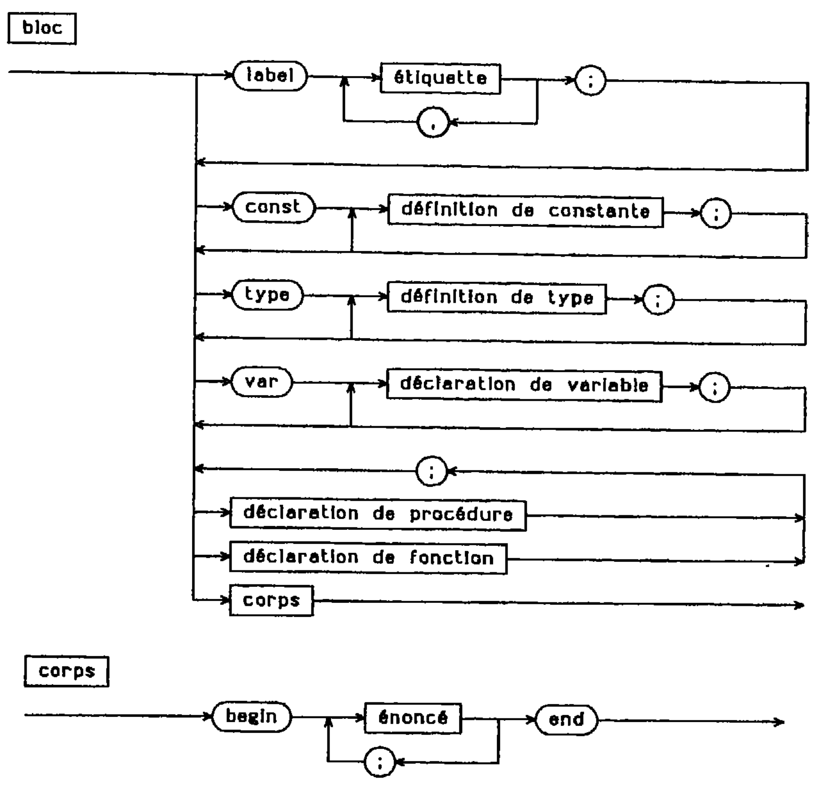
2 ـ قسم تعريفات الثوابت (أنظر 1.3.2)

3 ـ قسم تعريفات الأنواع (أنظر 4.3.2)

4 ـ قسم تصريحات المتغيرات (أنظر 2.3.2) .

5 ـ قسم تصريحات الإجراءات والدوال (أنظر 4.3) .





(programme : برنامج ؛ en-tête : عنوان ؛ bloc : فدرة ؛ identificateur : معرّف ؛ étiquette : label : وسم ؛ déclaration : تصريح ؛ définition : تعريف ؛ type : نوع ؛ variable : متغيّر ؛ procédure : إجراء ؛ fonction : دالّة ؛ corps : جسم ؛ énoncé : تعبير) .

إذا تفحصنا مخططات النحو هذه ، يمكن التحقق من كتابة البرنامج التالي :

البرنامج

```
program S2(input,output);
{ مجموع عددين }
var a,b,S:real;
begin read(a,b); S:=a+b; writeln(S) end.
```

- تم إحترام ترتيب مختلف الأقسام .
- الأقسام المدخلة من قبل الكلمات الدليلية label ، const ، type هي غير إلزاميّة .
- في نهاية البرنامج ، في «.end» ، تدخل «end» ضمن إطار الجسم بينما « . » لا تدخل .

سيتم غالباً في ما بعد إعادة إستعمال هـذه المخططات ؛ بـالأخص فإن تصـريح الإجراء أو الدالة يتضمن فدرة .

فيـما يخص التحليل ، فـإن الجسم يتـوافق مـع تعـريفـات التحليـل ، وأقسـام التصريحات تتوافق مع المعجم . أما الإجراءات والدوال فسيتم إخراجها عند الضرورة من جداول ثانوية .

عنوان البرنامج (L'en-tête du programme)

يحدد عنوان البرنامج الإسم المعطى للبرنامج ، إضافة إلى الأدوات الخارجية عن البرنامج التي سيتم استعمالها .

تسمح قائمة المعرِّفين بين المزدوجات بالعمل على التطابق ما بين أدوات داخلية في البرنامج (مسندة بواسطة معرفيها) وأدوات خارجية . فمثلاً

```
program nénant (input, output);
```

يجري إتصالاً بين أعضاء الإدخال والإخراج التابعة للحاسوب ، وبين السجلات ذات الإسم input وoutput المستعملة ضمنياً في البرنامج في تعابير القراءة (read) والكتابة (writeln) .

إن المعرّف néant الذي يلي الكلمة الدليليّـة Program ليس له مدلول في داخل البرنامج .

الفدرة (Le bloc)

إنها تصف أدوات (وسم ، ثابت ، نوع ، متغير) وأفعال (تعابير) . إنها تحمل إسماً محدداً بالرأس السابق للبرنامج إذا كان المقصود فدرة من البرنامج ، أو بعنوان الإجراء (أو الدالة) إذا كان المقصود فدرة من الإجراء (أو الدالة) .

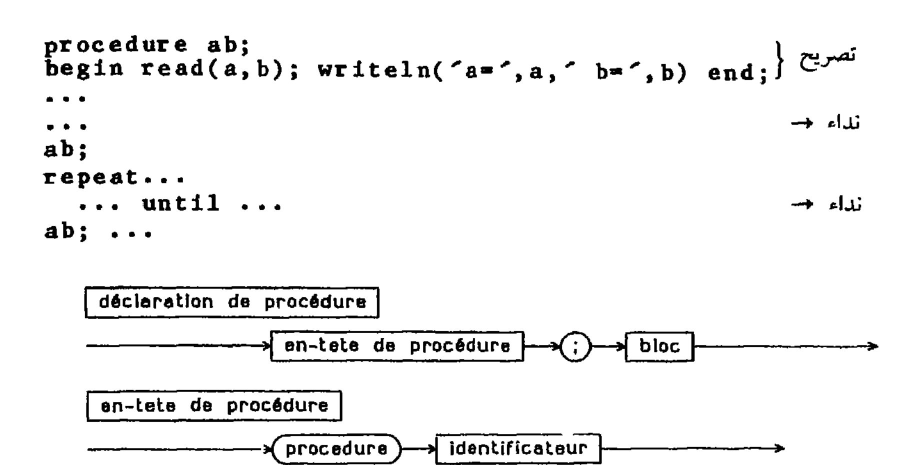
إجراء أو دالّـة ، دون وسيط (Procédure on fonction, sans paramètre)

إن عملية التصريح عن الإجراء أو الدالة تسمح بإعطاء إسم إلى جزء من البرنامج . بذلك فإنه من غير الضروري في ما يلي ، إعادة كتابة هذه التعابير عند كل إستعمال ، بل نكتفي بنداء الإجراء (أو الدالة) عن طريق تسميته :

```
... read(a,b); writeln('a=',a,'  b=',b);
    repeat ...
      ... until ...
    read(a,b); writeln('a=',a,'  b=',b); ...
```

يمكن كتابة الفقرة السابقة بشكل أسهل كما يلي :

```
procedure ab;
begin read(a,b); writeln('a=',a,' b=',b) end;}   تصريح
...
...                                              نداء ←
ab;
repeat...
  ... until ...                                  نداء ←
ab; ...
```



لقد تم إعطاء نحو الفدرة سابقاً . يمكن للفدرة أن تحتوي على تصريحات للإجراءات . يحتوي تصريح الإجراء على فدرة ، التي بدورها تحتوي على تصريحات لإجراءات . . . الخ . تسمى قاعدة النحو الخاص بالفدرة بالتكرارية .

3.2 ـ الأدوات المعالَجة

0.3.2 ـ أنواع (types)

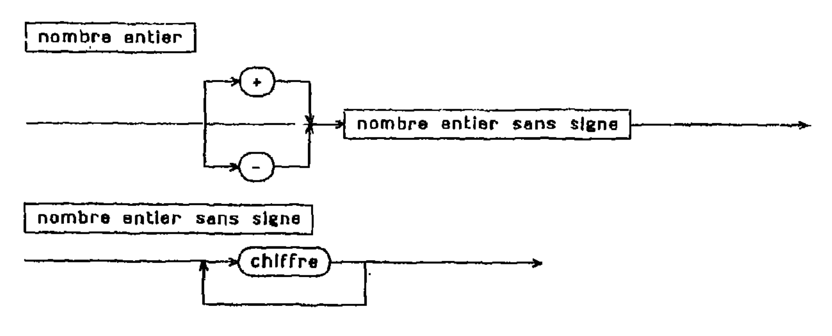
يجري الجسم حساباً على أدوات : ثوابت ومتغيرات . تتحرك قيمة المتغير في إطار محدد بنوع . يوجد أنواع بسيطة ، أخرى محددة مسبقاً (صحيح ، حقيقي ، سمة ، بولي) ، البعض الآخر مُنشأ من قبل المبرمج (فترة ، عدّ) ؛ يوجد أيضاً أنواع مركبة ، مُنشأة من قبل المبرمج إنطلاقاً من أنواع بسيطة ، مثل الجداول ، أو محددة مسبقاً ، مثل النصوص . بعض التركيبات الأخرى الأقل إستعمالاً ، سيتم بحثها في الفصل 4 : مجموعات ، دلائل وسجلات .

إنتبه : لا يمكن ، في تعبير ، في تعيين أو في مقارنة ، معالجة أدوات مجتمعة إلا إذا كانت ذوات أنواع مُتساوقة ؛ سيتم بحث هذه الفكرة الصعبة بعض الشيء في آخر الفقرة .

1.3.2 ـ ثوابت (Constantes)

لا يستطيع البرنامج تغيير أي ثابت ؛ يمكن أن يكون الثابت عدداً ، سمة ، سلسالاً أو معرفاً لثابت .

يخضع **العدد الصحيح** ، عنصر من النوع integer ، إلى النحو :



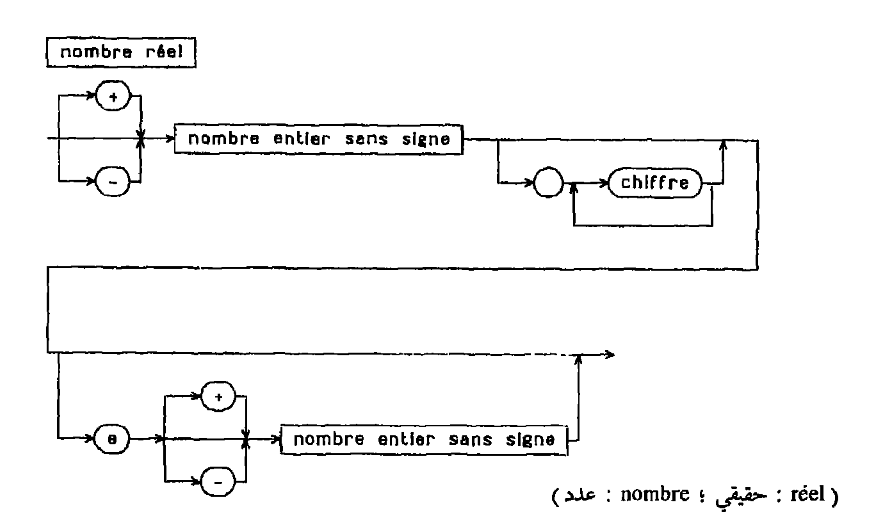
(chiffre : عدد ؛ signe : علامة ؛ entier : صحيح)

وذلك ضمن التمثيل في القاعدة 10 .

مثال : 4 3 – 0

لكن هناك نهاية ، معروفة بالثابت المحدد مسبقاً maxint ، لقيمة العدد الصحيح ؛ إن عدداً صحيحاً دون علامة ينتمي إلى الفترة 0... maxint .

يتوافق **العدد الحقيقي** ، عنصر من النوع real ، مع قيمة غير صحيحة ، ويمكن كتابته بشكل ثابت ، مثل 3.14 ، أو بشكل طليق الفاصلة ، مثل 0.314E1 حيث تعني «E» « مضروباً بـ 10 مرفوعاً إلى قوة » : 0.314×10^1 (النقطة الأنكلوسكسونية تحل مكان الفاصلة الفرنسية) .



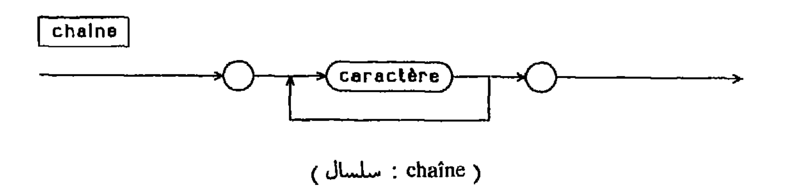
(réel : حقيقي ؛ nombre : عدد)

مثال : 3.14159   27.9E − 17   −5E−2   0.1E7   0.1e7

إنتبه : .1 و1. هم كتابات غير صحيحة

1.0   0.001E3 و10.0000E − 1 تعني نفس العدد

إن سلسالاً من n سمات هو متسلسلة من n سمات ، مأخوذة ضمن إطار لعب خاص بكل حاسب آلي . إنه العنصر الوحيد في اللغة (مع الملاحظات) الذي يلعب فيه التمثيل الحرفي دوراً مهماً . ننوّط السلسال عن طريق إحاطته بعلامات حذف (التي لا تدخل ضمن نطاق السلسال) ومن المتعارف عليه أن يتم تكرير علامة الحذف عند وجودها في سلسال : مثلاً السلسال aujourd'hui ينوط على الشكل التالي 'aujourd''hui' .

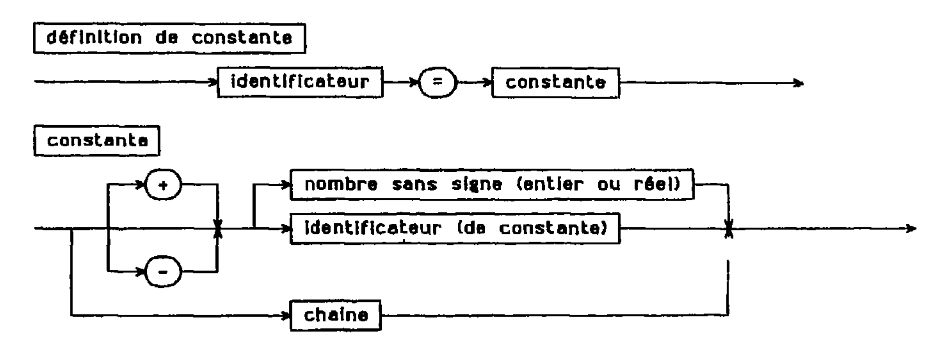


(سلسال : chaîne)

إن سلسالاً من سمة واحدة هو قيمة من النوع المحدد مسبقاً char (سمة) .

تعريف الثابت

إنه عملية تصريح ، مُدخلة بواسطة الكلمة الدليلية ,const التي تربط معرفاً بقيمة . فيما يلي ، يكون المعرف الذي تم تعريفه ، كناية عن تنويط آخر للثابت .



(définition de constante : تعريف الثابت identificateur : معرف) .

مثال :

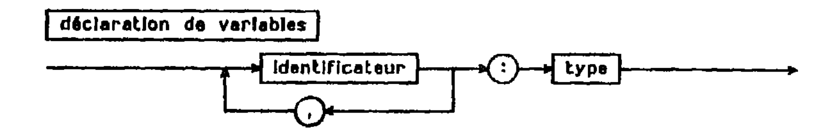
؛ ' ' = blanc (فراغ) N = 3 ؛ const e = 2.71828

بالعكس N + 1 .004 N1 هي كتابة غير مشرّعة .

2.3.2 ـ متغيرات

بعكس الثابت ، فإن المتغير هو كناية عن كيان تعيّـن له قيمة ، يجب التصريح عن كل متغيّـر ، وذلك بتحديد المعرِّف الذي يُسَمّيه ، ونوع القيمة الممكن تعيينها .

تتم عملية تصريح المتغيرات عن طريق إدخال الكلمة الدليلية **var** :



مثال :

Var i : integer ; (المعرّف للمتغير i هو من النوع integer (صحيح)

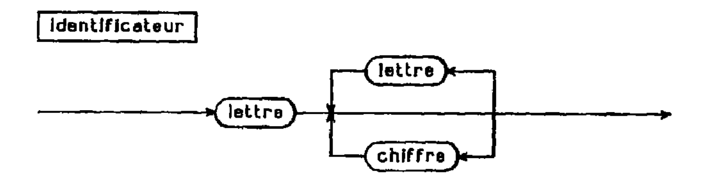
x, y, z: real ; (المعرّفين لـ z, y, x هم من النوع real (حقيقي)

a, b : T هي كتابة مختصرة لـ b: T ; a: T

لا يمكن إستخدام قيمة متغيّـرٍ إلا إذا تم تحديدها : إن العملية الأولى التي تتم على متغير ما في برنامج ما هي بالضرورة عملية تعيين (أو عمليـة مشابهة كالقراءة read) .

المعرّف

إن جميع السمات الخاصة بمعرف هي ذات مدلول :



(identificateur : معرف ؛ lettre : حرف ؛ chiffre : رقم)

لا يمكن لأي معرّفٍ أن تكون لديه نفس الكتابة الخاصة بكلمة دليلية (أنظر الملحق 3) ؛ بعض المعرفين المسمّين Prédéfinis (محددين مسبقاً) (مثل read ، maxint ، integer) لهم مسبقاً مدلولاً خاصاً ، هذا لا يمنع من إمكانية إعادة تحديدهم .

مثال : *Chapitre3* *X3* *heure*

بالعكس *var* *un_peu* *d'accord* *Chapitre3.2* *3X*

ليسوا معرّفين أصحاء .

ملاحظة : يوجد بعض النظم التي تحدد عدد السمات ذات المدلول بـ 10 (CDC) أو بـ 8 (USCD)

المظاهر الساكنة والمتحركة

إن التعيين u := u + 1 ، حيث تمت زيادة قيمة المتغير u ، يعرض نظرة تحريكية (dynamique) : الأداة u المعيّـنة ليست نفسها أداة الإنطلاق ، إنها تأتي بعدها في تسلسل الوقت . بالفعل فإن المقصود هو عملية حسابيـة على متسلسلة $u_i = u_{i-1} + 1$ حيث أن الدليل i يكون رقم ترتيب التغييرات المجراة على المتغير u ؛ هذا مما ننوّطه بـ u = u + 1 بالطريقة الساكنة (statique) في التحليل الإستنتاجي . هذا المظهر المتحرك هو أساسي في عملية فهم فكرة المتغير ؛ من الأفضل التفكير بطريقة سكونية بدل أن نعالج مباشرة أدوات متحركة .

3.3.2 ـ أنواع بسيطة

يُحدِّد النوع البسيط مجموعة منظّـمة من القيم . بعض الأنواع ، المسماة أوليّـة ، هي محددة مسبقاً (كانت موجودة سابقاً في البرنامج) : صحيح ، حقيقي ، بولي وسمة ، بعضها الآخر يتم تحديده بواسطة البرنامج الذي يستعملهـا : أنواع فترات وتعداد .

النوع حقيقي : real

تؤلف القيم مجموعة ثانوية ، محددة تبعاً لكل حاسب آلي ، من مجموعـة الأعداد الحقيقية

ملاحظة : تبعاً للحاسب المستعمل ، تتنوع دقة الأعداد الحقيقية ؛ فمثلاً بتمثيل « على 32 بتة » ، يجب أن تكون القيمة المطلقة للأسّ أقل من 37 ، وأن يكـون للجزء العشري ستة أرقام فقط (في القاعدة 10) ذات مدلول :

$$10^{37} < r < -10^{-37} \text{ أو } 10^{-37} < r < 10^{37}$$ (« على 32 بتة »)

فمثلاً 1.000 000 و1.0000 0001 هما بالنسبة للحاسب نفس الرقم . أما المؤثرات المطبقة على النوع الصحيح فهي + ، - ، * ، / .

النوع صحيح : Integer (نوع ترتيبي (ordinal))

تكونُ القيم مجموعة ثانوية من الأعـداد الصحيحة ، محـدّدة بالفتـرة .. maxint. maxint ـ، حيث أن maxint هي ثابتة محدّدة مسبقاً ومتعلقة بنوعية الحاسب المستعمل .

ملاحظة : على حاسب « ذو كلمات من 16 بتة » تكُون maxint = 32767 ، بينما إذا كان « ذو كلمات من 32 بتة » فإن maxint = 107321823 .

إن المؤثرات + ، − ، * ، div mod تطبَّـق على النوع صحيح . إن النوع صحيح هو نوع ترتيبي ، مثل الأنواع بولي ، سمة ، تعداد (وفترات) : يمكن أن نطبّـق على نوع ترتيبي ، الدوال succ, pred وord التي تعطي تباعاً السلف ، الخلف والعدد الترتيبي المُشرَك بقيمة .

النوع بولي : booléan (نوع ترتيبي)

إن القيم هي false (خطأ) وtrue (صح) ، إنهم معرّفين لثوابت محدّدة مسبقـاً حيث false تسبق true .

الخصائص :
ord (false) = 0 ord (true) = 1
succ (false) = true pred (true) = false
false < true

تُطبَّـق العمليات المنطقية العادية على النوع بولي : **and** (و) ، **or** (و) ، **not** (لا)
ملاحظة : تعني false باللغة الفرنسية «Faux» وtrue تعني «Vrai» .

النوع سمة : char (نوع ترتيبي)

إنه تعداد للُعْب السمـات (خاص بكـل آلة) ، التي يكـون للبعض منها تمثيـل تخـطيطي (أحـرف ، أرقام ، . . .) ولا للبعض الآخـر (سمات « ضبط » النقـل : RETURN ، ESCAPE ، . . .) .

تُحدَّد الأعداد الترتيبية للسمات بالتكويد المستعمل (ASCII ، EBCDIC ، BCD . . . انظر الملحق 1) ؛ إنها موجبة ، تبدأ من الصفر ، وهي متتالية .

الكتابة : بالنسبة للسمات « القابلة للطبع » : نضعها بين علامات حذف
مثال : 'A' '2' '—'

فيما خص السمات « غير القابلة للطبع » : لا يوجد تنويط محسوب (لكن الدالة chr تسمح بتخطي الصعوبة) .

إنه نوع ترتيبي ، فإذن تطبق عليه الدوال succ, pred وord ؛ نشير إلى أن لدى ord دالة مُعاكِسة ، تنوّط chr : chr (ord (c)) = c .

بالنسبة لترتيب السمات ، فإننا لا نعرف إلا الخصائص التالية :

1 ـ تكون الأرقام منظمة ومتلاصقة :

succ ('0') = '1' .. '0' < '1' < ... < '9'

2 ـ تكون الأحرف الكبيرة ، إذا كانت موجودة في الحاسب الآلي ، منظمة ، لكن ليس بالضرورة متلاصقة :

'A' < 'B' < ... < 'Z'

3 ـ نفس الخاصة بالنسبة للأحرف الصغيرة في حال وجودها في الحاسب .

$$'a' < 'b' < ... < 'z'$$

4 ـ تعطي مقارنة سمتين نفس نتيجة مقارنة أعدادهن الترتيبية .

$$c < d \Leftrightarrow ord\ (c) < ord\ (d)$$

إننا لا نعرف إذن أي شيء ذي صفة عامة حول ترتيب أيّ سمتين (العلاقة ما بين الأرقام والأحرف ، ما بين تمثيلين مختلفين للأحرف ، موضع التباعد (' ') في اللعبة ، . .) : تختلف هذه العلاقات من حاسب إلى آخر .

الأنواع تعداد (نوع ترتيبي) (Types énumérés)

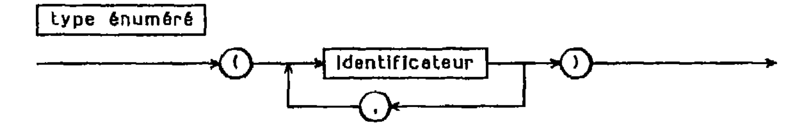
إنه مجموعة منظمة من القيم ، المُحدّدة بالتعداد للمعرّفين الذين يعبّـروا عن هذه القيم . مثال :

var Jour : (lundi, mardi, mercredi , jeudi, vendredi)

نهار الإثنين ، الثلاثاء ، الأربعاء ، الخميس ، الجمعة

يمكن أن يأخذ المتغير Jour فقط إحدى القيم من Lundi حتى vendredi ويمكننا كتابة jour : = succ (jour) ، مع الإشارة بأن lundi ليس له سلف وvendredi ليس له خلف .

تكون الأعداد الترتيبية متتالية وتبدأ من الصفر :



(type énuméré : نوع تعداد ؛ identificateur : معرّف)

كما يظهر فإن لهذه الأنواع تعداد فائدة كبيرة : إنها تغني عن تكويدات مملّـة التي هي بالأغلب مصادر للأخطاء .

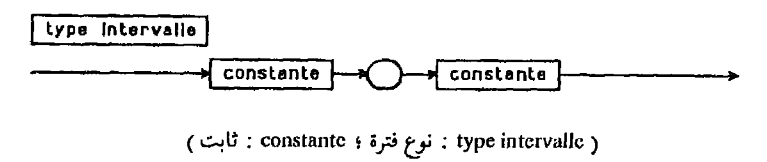
الأنواع فترات (ترتيبية) (type intervalles)

إنه يحدّد فترة من القيم المتتاليـة ، المأخـوذة في نوع تـرتيبي ، يسمى نوع سـائد (hôte) ؛ تنتمي الحدود الدنيا والقصوى المستعملة إلى الفترة التي ترث العمليات المطبّـقة على النوع سائد (host) .

مثال :

1... 100   − 10 ... + 10   '0' ... '9'

لا يمكن لمتغير مصرح من داخل نوع فترة أن يأخذ قيمةً خارج هذه الفترة . سنجد هذه الخاصة عند دلائل الجداول .

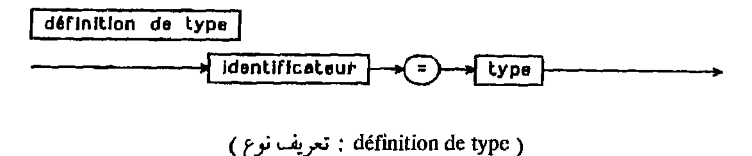


(ثابت : constante ؛ نوع فترة : type intervalle)

مثال : مع معرفة النوع (lundi, mardi, mercredi, jeudi, vendredi) ، يمكن تكوين الأنواع lundi ... jeudi أو mardi ... jeudi .

4.3.2 ـ تعريف نوع

يسمح تعريف نوع بربط إسمٍ (معرِّف) بنوع . تُدخل هذه التعريفات بواسطة الكلمة الدليلية **type** .



(تعريف نوع : définition de type)

هكذا يمكن إستعمال الإسم المعطى إلى النوع في كل مرة نحتاج فيها إلى وصف النوع : لإنشاء نوع مركب ، في تصريح متغيرات ، . . .

مثال :

```
type reponse=(oui,non,peutEtre);
     chiffre='0'..'9';
     typeSimple=(reel,entier,booleen,caractere);
     typeOrdinal=entrer..caractere;
```

(ترتيبي : ordinal ؛ بسيط : simple ؛ رقم : chiffre ؛ جواب : réponse)

فإذن : **var** dif: chiffre ، تحل مكان **var** dig : '0' ... '9' ;

يكون النوع البولي محدد مسبقاً بشكل منسّق وخارج إطار البرنامج وذلك بواسطة **type** boolean = (false, true); .

تعريف واحد

كل معرِّف ، بعد أن تم تصريحه أو تعريفه ، لا يمكن أن يعُد نفسه في نفس النص .
هكذا ومع التعريفات السابقة ، فإنه من غير الممكن كتابة :
var réel : real; (سيعني ذلك إعادة تعريف réel)

أو ; var scalaire: (booléen , énuméré) (سيعني ذلك إعادة تعريف booléen) .

ملاحظة : هذه القاعدة لا تطبق بتاتاً بمجرد أن تغيرت القدرة .

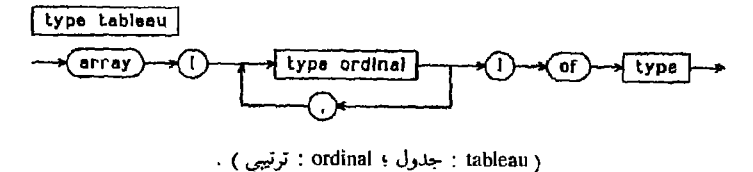
5.3.2 ـ تكوين الجدول

الجدول كناية عن مجموعة من المركّبات جميعها من نفس النوع ، ولها عدد ثابت معروف مسبقاً . يرتبط عدد المركّبات بفترة تغيّر الدليل .

التصريح : نوع المركّبات **of** [نوع الدليل] **array**

ـ لا يمكن أن يكون نوع الدليل صحيحاً (لكن يمكن أن يكون فترة) ولا حقيقياً ؛ يجب أن يكون لديه نوعاً بسيطاً ،

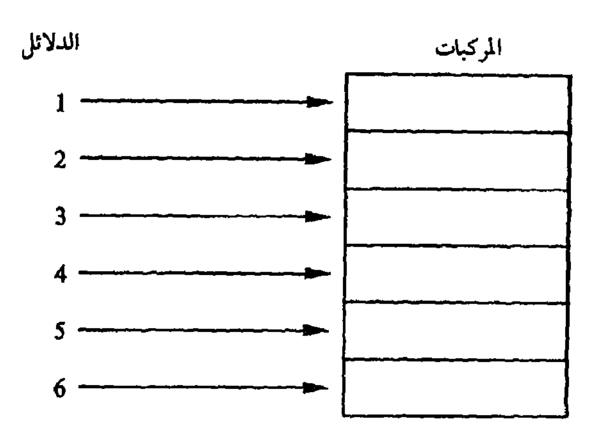
ـ إن نوع المركّبات هو غير محدد (أيّ كان) ، يمكن أن يكون جدولاً .



(tableau : جدول ؛ ordinal : ترتيبي) .

إحداثيات نقطة في R^6 : **array** [1...6] **of** real

تتوافق فكرة الجدول مع تطبيق لكل قيمة من نوع الدليل على مركّب مختلف :



نستعمل الجدول عندما نريد تخزين قيمٍ ذوات عدد محدد وكلها من نفس النوع : علامات إمتحان ، n كشف للحرارة ، . . .

يمكن أن يكون نوع الدليل

- فترة

```
array [0..99] of integer
array ['A'..'Z'] of real
```

- تعداد

نترون　　بروتون　　الكترون

```
array [(neutron,proton,electron)] of...
```

- نوع معرّف سابقاً

```
type reponse=(oui,non,peutEtre);
     chiffre='0'..'9';
     T1=array [reponse] of integer;
var T2: array [chiffre] of boolian;
    T3: array [boolean] of real;
    T4: array [char] of T1;
```

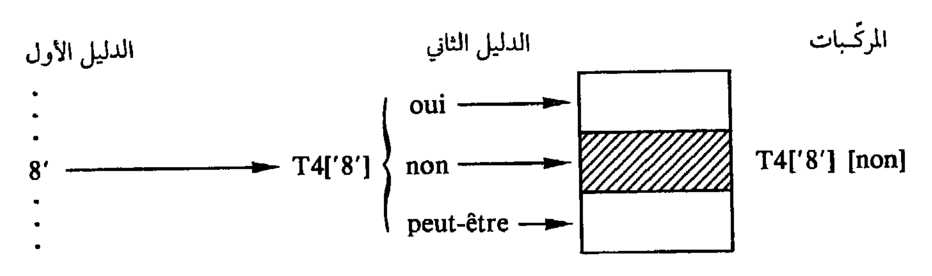
(oui : نعم ؛ non : كلا ؛ peut Etre : ربما)

نستعمل مركّباً لمتغير من نوع جدول عن طريق تدوين دليله بين معقّفين

مثال : آخذين بعين الإعتبار التعريفات السابقة ، يمكن كتابة :

```
T2['5']:=true;
T3[true]:=0.0;       T3[T2['5']]:=0.0;
T4['8'][non]:=109;
```

T4 هو جدول من جدول (نسميه جدولاً ذي بُعدَيْن) :



T4 هو من النوع **array** *[char]* **of array** *[réponse]* **of** *integer*

T4 ['8'] من النوع **array** *[réponse]* **of** *integer*

T4 ['8'] [non] من النوع *integer*

إختصارات :

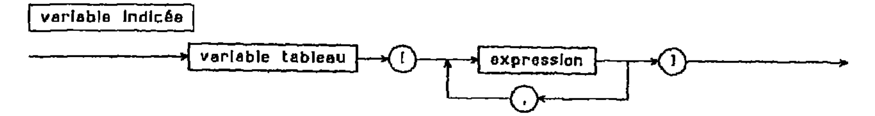
1 ـ في حال كان مركّب الجدول جدولاً ، يمكننا عند كتابة المتغيّر الدليلي ، استبدال] [بـ ,

مثال : T4 ['8', non] هو مكافىء لـ T4 ['8'] [non]

2 ـ يمكننا ، في خلال عملية تعريف النوع ، إختصار [**of array**] إلى ,

مثال : array [char] of array [reponse] of integer

هو مكافىء لـ array [char, reponse] of integer



(variable indicée : متغير دليلي ؛ tableau : جدول ؛ expression : تعبير)

أمثلة : مصفوفة 10 × 10 :

M : array [1..10, 1.. 10] of real;

ـ كمية المعادن الموجودة في الجُزُر من خلال 20 عملية سبر ، مع 30 جزرة في كل عملية :

array [(Fe, Cu, As, Ag, E), 1...20, 1...30] of real;

ـ جداول القيم لدالّـة عُقديّـة من متغيرين صحيحين :

f: array [0..100, 0..10] of

array [(reel, imaginaire)] of real;

(réel : حقيقي ؛ imaginaire : خيالي)

ـ نتيجة سبرٍ للرأي العام مؤلّـف من 60 سؤالاً :

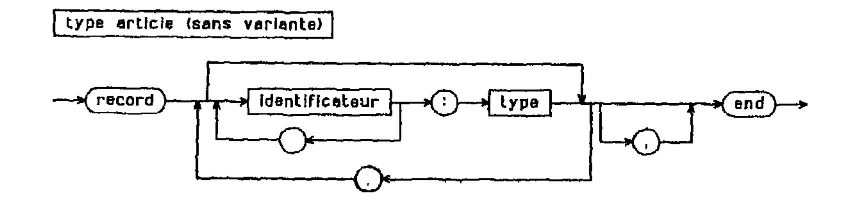
array [1..60] of (oui, non, peut Etre);

(سنرى النوع سلسال في الفقرة 3.4) .

6.3.2 ـ تكوين الفقرة (article)

تكُون الفقرة مجموعة من مركّـبات ذوات نوع غير محدد (أيّ كان) ، ولها عدد محدّد مسبقاً .

يمكن أن تكون هذه المركبات من أنواع مختلفة ولـذا فإنـه لا يمكن بلوغها بـدليل موحّـد ، بل بإسم ، يسمى مركب الفقرة بالحقل .



مثال :

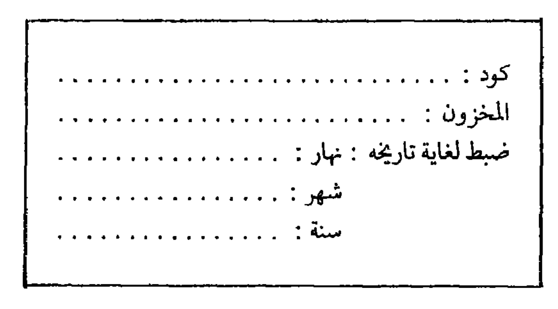
```
type date= record
            jour:1..31; mois:1..12; an:1840..1999
          end;
     modele= record
              code:0..9999; enStock:integer;
              miseAjour:date
            end;
var stock: array[1..200] of modele;
```

(record : قياس ؛ date : تاريخ ؛ modéle : نموذج ؛ stock : مخزون ؛ jour : نهار ؛ mois : شهر ؛ an : سنة ؛ mise A jour : ضبط لغاية تاريخه ؛ code : كود ؛ variante : مشتق) .

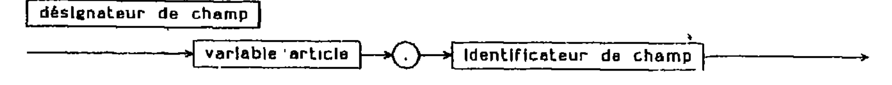
بالحدس ، يمكن التفكير بالفقرة كما لو أنها كانت « سجلاً » من الكرتون :

مخزون : مجموعة من 200 سجلاً من النموذج

نموذج :



يُكْتب البلوغ في متغير فقرة على شكل مؤشّر للحقل :



(désignateur de champ : مؤشر للحقل ؛ variable article : متغيّر فقرة)

مثال : stock [i]	هو متغير فقرة ؛ فإذن :
stock [i] .Code	هو من النوع 0..9999
stock [i] . en stock	هو من النوع integer
stock [i] . mise A jour	هو متغير فقرة
stock [i] .mise A jour. jour	هو من النوع 31 ..1

تُسْتعمل الفقرة عندما يجب تخزين قيم من نوع غير محدد وبعدد معروف .
ملاحظة : سيتم شرح مشتقات الفقرة في 5.4 .

الفقرة والجدول (Article et tableau)

كما رأينا ، فإنه تتشابه كثيراً فكرة الفقرة والجدول ؛ فالمقصود في الحالتين هو مجموعة من المتغيرات ذوات عدد ثابت .

هكذا ، يمكن تمثيل عدد عقديّ بواسطة

جدول : **Var** plext: **array** [(réel, imaginaire)] **of** real;

أو فقرة : **Var** plexa: **record** réel, imaginaire: real **end;**

هذا ما يؤدي الى كتابة [plext [réel و plexa . réel ؛ يتم الإختيار تبعاً للإستعمال الذي يجريه برنامج الأدوات العقدية .

ملاحظة : فيما خصّ الحاسب ، تمثل الفقرات والجداول بنفس الطريقة داخل الذاكرة ؛ البلوغ يختلف :

ـ بالنسبة للجدول فإن عنوان [T [x = عنوان أوّل عنصر من T + (x - الحد الأدنى) * حجم المركّب .

ـ بالنسبة للفقرة فإن عنوان A.x = عنوان أوّل عنصر من A + إنتقال الحقل x في الفقرة (قيمة معروفة قبل تنفيذ البرنامج) .

كما نرى فإن بلوغ حقلٍ من الفقرة هو أسرع من بلوغ مكوّنٍ من الجدول ، خصوصاً عندما يكون جدولاً ذي عدة أبعاد :

var *T* : **array** *[a..b, c..d, e..f]* **of** *ty*
alors *adresse de T [x, y, z] = adresse T [a, c, e] +*
(x − a) ★ (d − c + 1) ★ (f − e + 1) ★ taille d'un composant +
(y − c) ★ (f − e + 1) ★ taille d'un composant +
(z − e) ★ taille d'un composant

(adresse : عنوان ؛ taille d'un composant : حجم المركّب)

7.3.2 ـ قواعد التسَاوُق (compatibilité)

يكون النوعان T1 وT2 متساوقين ، إذا كان لديهما إحدى هذه الخصائص :

أ ـ T1 وT2 هما نفس النوع (لديهم نفس الإسم)

ب ـ الواحد هو « فترة » من الآخر ، أو الإثنين معاً هما « فترات » من نفس النوع .

ج ـ هما نوعا مجموعات منشآن على أنواع أساسية متساوقة ؛ T1 وT2 كلاهما معلّبين ، أولاً .

د ـ هما نوعا متسلسلات لديهما نفس عدد المركّبات .

هكـذا وبعكس الحدس ، فـإن نوعـين لديهـا نفس التكوين ، ليسـا بالضـرورة متساوقين :

```
type T1 : record x, y : array [ 0..1 ] of char end;
Var a : rdcord x, y : array [ 0..1 ] of char end;
    b : T1
```

بل يجب أن نكتب :

```
Var a : T1 ;
    b : T1 ;
```

فيما يلي سنستعمل كثيراً قواعد التساوق هذه خصـوصاً قـواعد التسـاوق المتعلقة بالتعيين (1.3) .

4.2 ـ الدَّخْل ـ الخَرْج : (OUTPUT-INPUT)

السجل هو كناية عن متسلسلة مركّبات كلهـا من نفس النوع وبعـدد غير محـدد (بعكس الجدول) ؛ النص هو كناية عن سجل سمات ، مهيكل بشكل أسطر .

يتم التعريف المسبّـق لنصّين في برنامج باسكال :

input ، نص معطيات ، يتم بلوغه بالقـراءة بواسـطة الإجراء read (إقـرأ) ، وoutput ، نص نتائج ، يتم بلوغه بالكتابة بواسطة الإجراء write (اكتب) . تتوافق هذه النصوص مع :

	إستعمال متحرك	إستعمال جامد
الدخل	ملامس	قارىء البطاقات
الخرج	الشاشة	الطابع

تكـون النصوص input وoutput ، في البـرنامـج وفي حال استعمـالهما ، أدوات خارجية ويجب أن تظهر كوسيط للتصريح **program;** وبذلك فإن علاقة ستنشأ عند التنفيذ بين الأداة الخارجية والأداة الداخلية والتي ستعالج بواسطة الإجراءات المعرّفة سابقاً read وwrite (وكذلك readln ، writeln ، eof . . .) .

النص input هو بالشأن **معاينة** inspection : لا يمكن إستعماله إلا للقراءة .

النص output هو بالشأن **تناتج** génération : لا يمكن استعماله إلا للكتابة مع العلم بأنه فارغ في البدء .

1.4.2 ـ نهاية السجل

بكل نوع سجل (نص مثلاً) نُشرِك في كل لحظة :

ـ متسلسلة من القيم

ـ موقع في المتسلسلة ؛ هذا الموقع يتطابق مع موضع النافذة

ـ شأن ، معاينة أو تناتج .

تسمح أوّليّـات النيل في السجل بإزاحة النافذة ضمن متسلسلة القيم ، بقيمة واحدة في كل مرة .

في الشأن معاينة («input» مثلاً) ، لا يكون النيل ممكناً إلا إذا لم تصل النافذة إلى نهاية السجل ، أي أنه ما زال يوجد مركّـبات مطلوب قراءتها .

في الشأن تناتج («output» مثلاً) ، لا يكون النيل ممكناً إلا إذا كانت النافذة في نهاية السجل ، أي أنه من الممكن توسيع السجل .

تسمح الدالة المعرّفة مسبقاً eof ، ذات النتيجة البولية ، بمعرفة قيمة هذا الشرط « نهاية السجل » (End Of File بالإنكليزية) ؛ عند كتابتها بدون وسيط ، تُطبَّـق eof على النص input .

eof (f) = true
والشأن تناتج } ⇐ كتابة (write) مُمْكِنة على السجل f

eof (f) = false
والشأن معاينة } ⇐ قراءة (read) مُمْكِنة على السجل f

مثال : قراءه لكل السمات الموجودة في السجل input ، مع متغيّـر C من النوع char (سمة) :

While not eof **do begin**
read (C) ; { إستعمال السمة المقروءة . . . } **end**

في البداية ، يكون لدينا «eof (output)» و «**not** eof (input)» (شرط أن لا يكون النص input فارغاً) .

2.4.2 ـ نهاية السطر (Fin de ligne)

النص هو كناية عن سجل من السمات ، مهيكل بشكل أسطر .

في الشأن **تناتج** ، نعرِّف نهاية السطر عن طريق إستعمال الإجراء المعرّف مسبقـاً writeln ؛ إنها الطريقة الوحيدة (مع page وrewrite) لتعريف نهاية السطر .

في الشأن معاينة ، تسمح الدالة المعرّفة مسبقاً eoln ، وعند كل موضع للنافذة ،

بمعرفة قيمة الشرط نهاية السطر (End Of Line بالإنكليزية) في حال أعطَت eoln القيمة true ، فإن السمة المقروءة هي تباعد (espace) . هذا المركّب الخاص « نهاية السطر » (مقروء كما لو أنه تباعد) لا يمكن تفرقته عن المركّب العادي « تباعد » إلاّ إذا استعملنا الدالة eoln .

مثال : مع المتغيّر c من النوع char والمحتوى التالي للسجل input :

```
PETITbAbPETIT,
bL'OISEAUbFAITb
bSONbNID.
```

(حيث أننا رمزنا إلى السمة تباعد بالحرف b)

1 ـ while not eof do begin read (c); write (c) end

تعطي النتيجة

PETITbAbPETIT,bbL'OISEAUbFAITbbbSONbNID.b

```
2) 1. while not eof do begin
   2.   while not eoln do begin
   3.     read(c); write(c) end;
   4.   read(c); write('!') end
```

تعطي النتيجة

PETITbAbPETIT,!bL'OISEAUbFAITb!bSONbNID.!

(تصبح eoln صح «true» للمرة الأولى عندما تعطي read (c) ، في السطر 3 ، السمة c = ',' ؛ بينما تسمح read (c) في السطر 4 « بامتصاص » التباعد المتوافق مع نهاية السطر ، الخ . .) .

إذا كانت eof صح فإن نداء eoln هو خطأ .

3.4.2 ـ الدّخل : readln, read

عند تطبيقه على نص ، في الشأن معاينة (مثل input) ، فإن الإجراء المعرَّف مسبقاً read يسمح بقراءة :

ـ سمة (تلك الموجودة في النافذة ، التي ستنتقل فيما بعد إلى السمة التالية) ؛

ـ عدد صحيح ، مع أو بدون علامة : يتم تجاهل التباعد ونهاية السطر السابقين للقيمة ؛ تتوقف القراءة فور عدم وجود سمة من العدد الصحيح في النافذة .

مثال : مع **Var a, b : char; i, j, k : integer**

والمعطيات «bb − 12b0 ★ 1 ».

فإن (i, j, a, k, b)

تعطي
i = − 12
j = 0
a = '★'
k = 1
b = '.'

ـ عدد حقيقي ، مع أو بدون علامة : يتم تجاهل التباعد ونهاية السطر السابقتين للقيمة ؛ تتوقف القراءة فور عدم وجود سمة من العدد الحقيقي في النافذة .

إختصارات :

read $(v_1, v_2,..., v_n)$ هو مكافىء لِـ
begin *read* (v_1); read (v_2);...; read (v_n) **end**

إنتبه : يجب أن تكون القيمة المقروءة متساوقة بالنسبة للتعيين مع المتغير المعيَّـن .

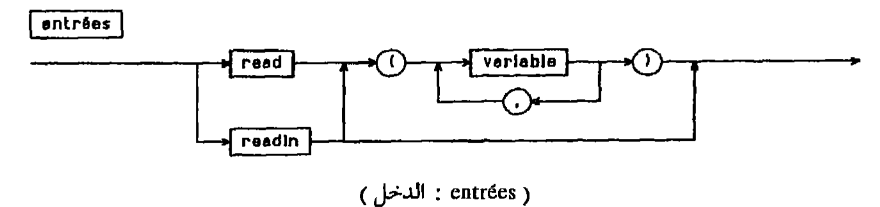
ملاحظة : لا يسمح Read بقراءة قيمة من النوع تعداد (بولّـي مثـلاً) ، دليل (Pointeur) ، مجموعة ، جدول ، سلسال أو سجل .

إن قراءة أعداد صحيحة أو حقيقية لا تسمح باستعمال الدالة eof ذلك لأنه يجري « تلبيس » سجل السمات بتكوين مختلف .

إن الإجراء المعرّف مسبقاً readln والمطبّـق فقط عـلى النص ، يعمل عـلى وضع النافذة على بداية السطر التالي (يعني إذن تخطّـي نهاية السطر) ، في حال وجوده .

إختصارات :

readln $(v_1,..., v_n)$ هو مكافىء لِـ
begin *read* (v_1);...; read (v_n); readln **end**



(الدخل : entrées)

4.4.2 ـ الخرج : writeln, write

عند تطبيقه على نص ، في الشأن تناتج (مثل output) ، فـإن الاجراء المعـرّف مسبقاً write يسمح بكتابة :

ـ **سمة** ؛ يعمل write (e) على كتابة قيمة e على شكل سمة (إنه write (e:1)) يعمل write (e:n) على كتابة n-1 تباعد ، ومن ثم قيمة e على شكل سمة .

مثال : write ('a', 'a': 2) يعطي aa

ـ **عدد صحيح** ؛ يعمل write (e) على كتابة التمثيل العشري لـ e (إنه write (e:n) حيث تتعلق n بنوعية الحاسب الآلي المستعمل) .

يعمل write (e:n) على كتابة التمثيل العشري لـ e ، مسبوقاً بقدر ما يلزم من التباعد للحصول في النهاية على n سمة . إذا تعدّى عدد الأرقام (زائد العلامة –) n ، يكون قد تم تجاوز حقل الكتابة .

مثال : write (12: 2, -12: 4, 127:1) يعطي 12- 12127

ـ **عدد حقيقي** ؛ يعمل write (e) على كتابة قيمة e على شكل عائم (إنه write (e:n) حيث تتعلق n بنوعية الحاسب الآلي المستعمل) .

يعمل write (e: n) على كتابة قيمة e على شكل عائم ، مسبوقاً بعدد من التباعد لملء n سمة بالإجمال . إذا كان حقل الـ n سمة غير كافياً ، فإنه سيتم توسيعه .

يعمل write (e:n:d) على كتابة قيمة e على شكل ثابت ، مع d رقم للقسم الكَسْري من الجزء العشري ؛ يسبق ذلك عدد من التباعد لملء n سمة بالإجمال . إذا كان حقل الـ n سمة غير كافياً ، فإنه سيتم توسيعه .

ـ **سلسال** ؛ يعمل write (e) على كتابة الـ x سمة من السلسال e وذلك بالترتيب .

يعمل write (e: n) على كتابة الـ n أول سمات من السلسال e ، مسبوقاً بقدر ما يلزم من التباعد للحصول على حقل من n سمة ، في حال x < n .

ـ **بولي** ، يعمل write (e) على كتابة السلسال 'true' أو السلسال 'false' (بالحرف الصغير أو الكبير) .

يعمل write (e: n) على توسيع الحقل إلى n سمة ، كما هي حال السلسال .

إختصارات : write (e_1, e_2, ... , e_n) هو مكافىء لـ

begin write (e_1); write (e_2); ...; write (e_n) **end**

ملاحظة : لا يسمح write بكتابة قيمة من النوع تعداد (ما عدا البولي) ، دليل (pointeur) ، مجموعة جدول، فقرة أو سجل .

إن الإجراء المعرّف مسبقاً writeln والمطبق فقط على النص ، يعمل على إنهاء السطر .

إختصارات : writeln (e_1, ..., e_n) هو مكافىء لـ

begin write (e_1); ...; write (e_n) ; writeln **end**

مثال : أكتب n نجمة على سطر

for x: = 1 to n do write (' * '); writeln

مثال : أكتب p * n خط وصل على سطر ، بمجموعات من p مفصولة بعلاقة جمع (+) :

--- + --- + --- +

```
for i:=1 to n do begin
  for j:=1 to p do write('-'); write('+') end;
writeln
```

مثال : أعد نسخ المعطيات (c ، متغير سمة)

```
while not eof do begin
  while not eoln do begin
    read(c); unite(c) end ;
  readln; writeln end
```

5.4.2 ـ ترتيب الصفحات أو الإخراج (mise en page)

يُسبِّب الإجراء المعرّف مسبقاً page بتخطي الصفحة ، عندما يكون سجل الخرج مطبوعاً على جهاز ضوئي (périphérique) ملائم .

ملاحظة : على بعض الحاسبات الآلية ، ومع بعض الأجهزة الضوئية للخرج ، تكون السمة الأولى لكل سطر غير مطبوعة ، لكن تفهَم كسمة تحكّم لتقديم الورق ؛ تطبّق إذن بشكل عام المصطلحات التالية :

' ' (تباعد) : إنتقال عادي إلى السطر .

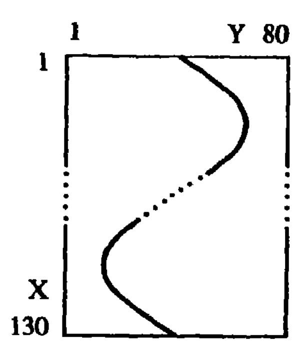
'0' (صفر) : أقفز سطراً قبل الطباعة .

'1' : قفز الى الصفحة قبل الطباعة .

'+' : خطوة تقديم الورق .

6.4.2 ـ مثال : مُنْحَنى

يُمكن الحصول على رسم تقريبي لمنحنى على الطابعة أو على الشاشة ؛ مثلاً لرسم y = sin(x) ، ضمن إطار من 80 عاموداً و130 سطراً ، نطبع



عندنا إذن العلاقات ، بين الإحداثيات X (سطر) ، Y (عامود) وx, y الإنطلاق

Y	1	→	80	X	1	→	130
py	− 1	→	+ 1	px	0	→	4 π environ

soit $Y = 1 + 79 \star (py + 1)/2$
$py = \sin(px)$
$px = 4\pi (X/130)$
X تتغير من 1 إلى 130

التعريفات	المعجم
نتيجة = أكتب « نقطة في العامود Y » لـ X من 1 إلى 130	ـ Y (1...80) لـ X : رقم العامود ـ X (1...130) : رقم السطر

لـ X ← Y

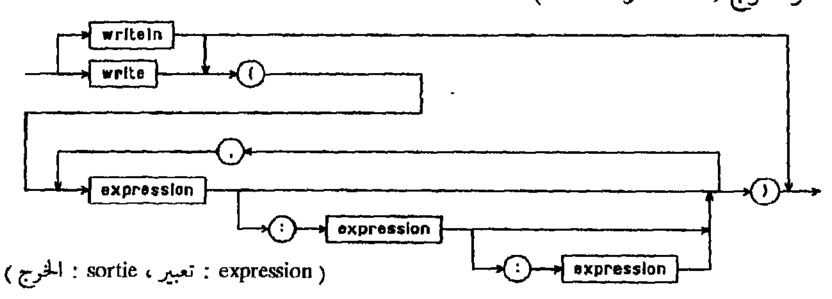
ـ py (حقيقي)	3	y = entier (1 + 79 * (py + 1) /2
ـ px (حقيقي)	2	py = sin (px)
ـ pi = 3.14159	1	px = 4 * pi * X / 130

```
program courbe(output);
{ ارسم منحنى الجيب على الشاشة أو الطابعة }
const pi=3.14159;
var Y:1..80;   { رقم العامود }
    X:1..130;  { رقم السطر } ;
    px,py:real;
begin
  for X:=1 to 130 do begin
    px:=4.0*pi*X/130.0;  py:=sin(px);
    Y:=trunc(1.0+79.0*(py+1.0)/2.0);
  writeln('.':Y) end
end.
```

(منحنى = courbe)

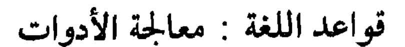
ملاحظة : إن معالجة السجلات سيتم شرحها بشكل أوفر في 7.4

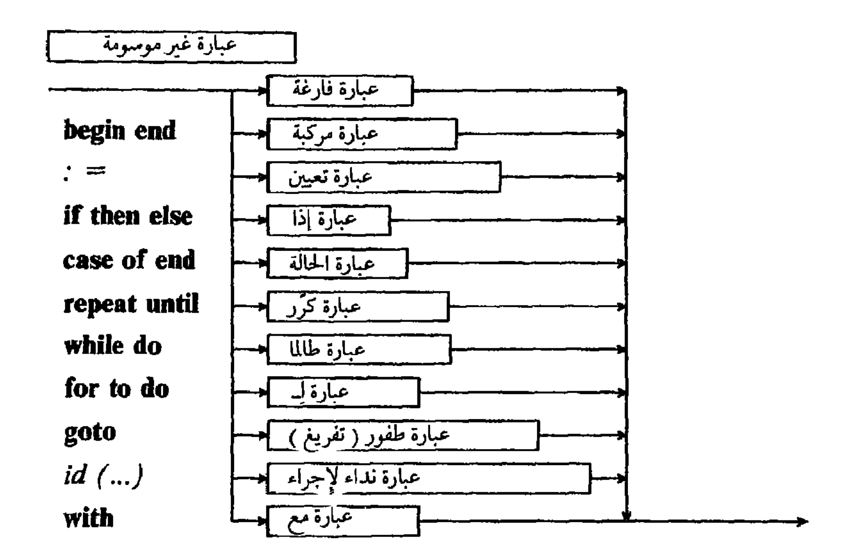
نحو الخرج (write أو writeln)



(الخرج : sortie ، تعبير : expression)

الفصل الثالث





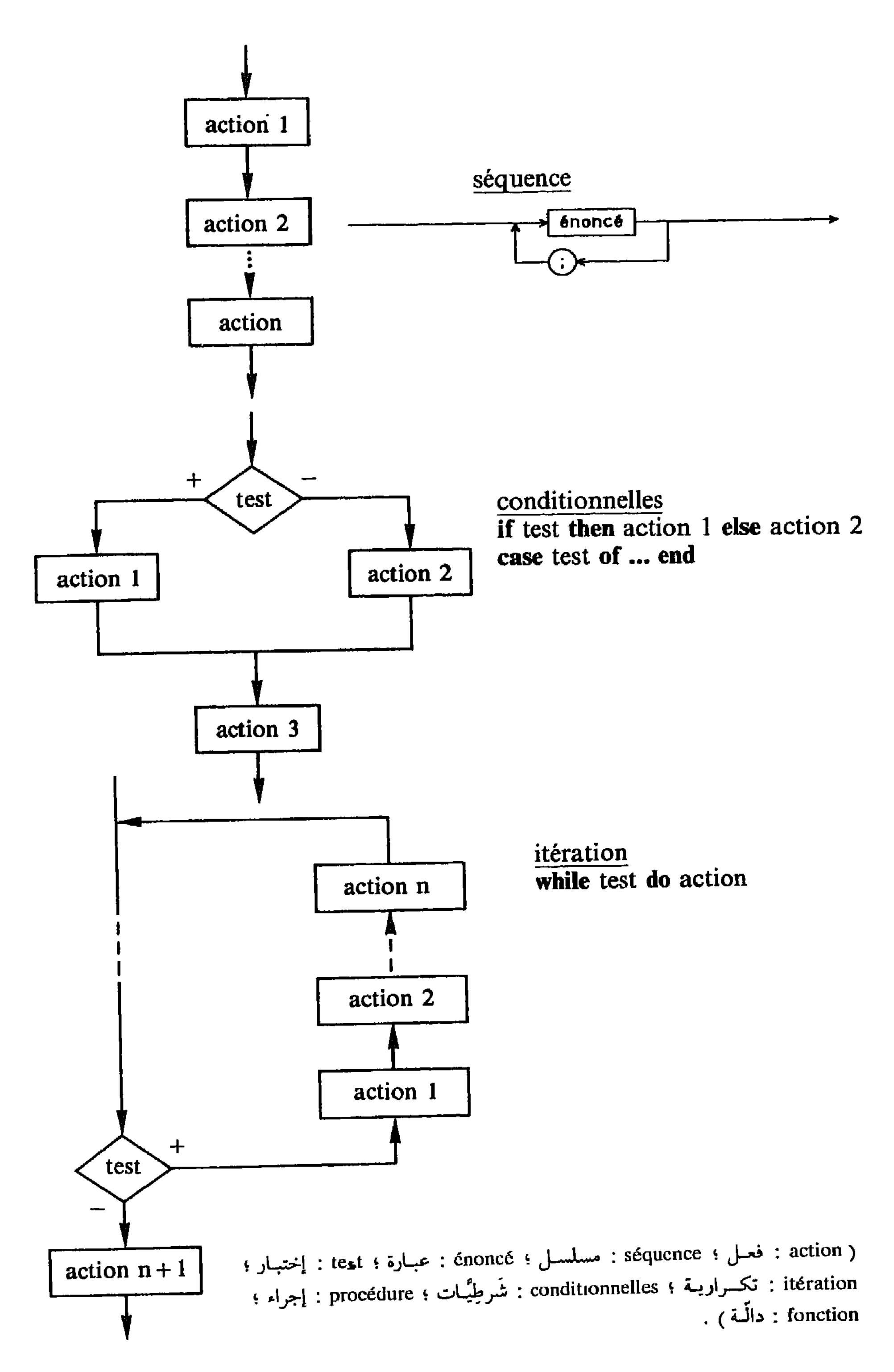
تتألف فدرة (bloc) برنامج من قسمين كبيرين :

ـ وصف **للأفعال** الواجب إتمامها ، مكتوبة على شكل **عبارات** ؛

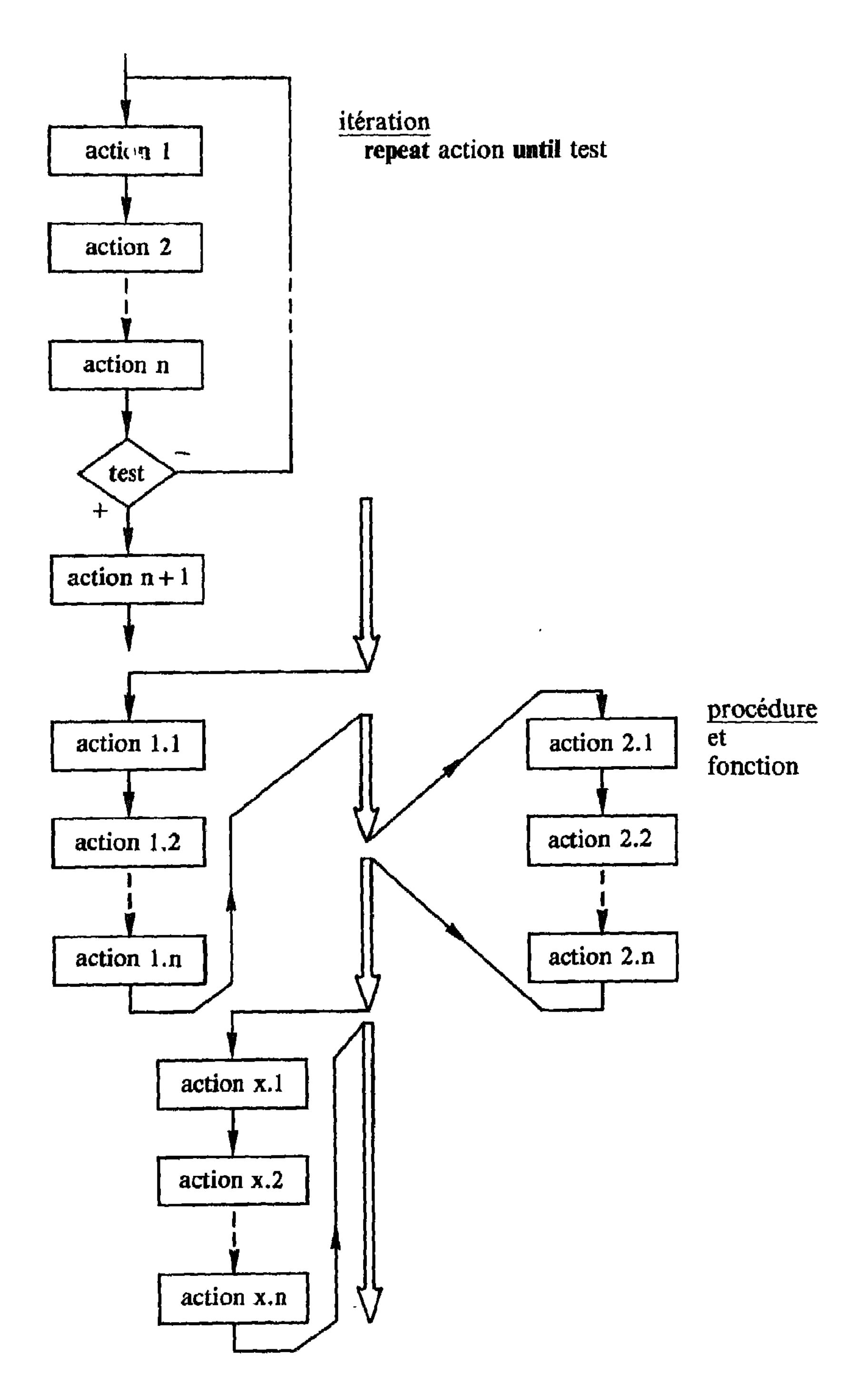
ـ وصف **للأدوات** المعالَجة من قِبَلِ هذه العبارات ، متمّم بواسطة **تصريحات وتعريفات** : تصريح متغيرات ، تعريف أنواع وثوابت .

0.3 ـ عبارات (Enoncés)

إن سيل الحسابات الخاص بخوارزم أو برنامج لا يَقِرُّ بالوصف الخطّي : يوجد حلقات ، رجوع إلى الوراء وقفزات . تحتوي لغة الباسكال على تركيبات تحكّم ، بأعداد صغيرة ، تسمح بإجراء هذا الوصف :

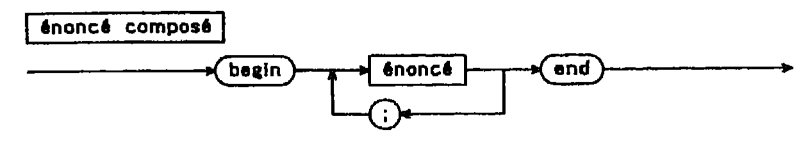


(action : فعل ؛ séquence : مسلسل ؛ énoncé : عبارة ؛ test : إختبار ؛ itération : تكرارية ؛ conditionnelles : شرطيات ؛ procédure : إجراء ؛ fonction : دالّة).



itération
repeat action **until** test
action 1
action 2
action n
test
−
+
action n + 1
action 1.1
action 1.2
action 1.n
procédure
et
fonction
action 2.1
action 2.2
action 2.n
action x.1
action x.2
action x.n

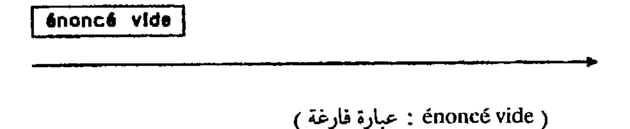
تُجمِّع العبارة المركبة مسلسل من العبارات



(énoncé composé : عبارة مركّبة) .

(إن الجسم هو كناية عن عبارة مركبة) .

يمكن إستعمال العبارة الفارغة في برنامج ، وبشكل عام بهدف التمكّن من الإدخال الحر لـ « ؛ » :



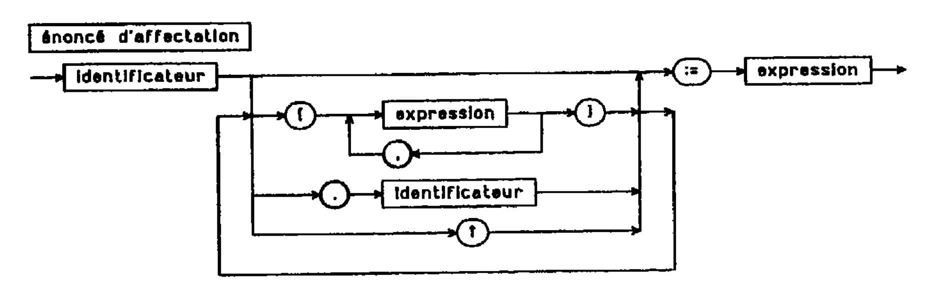
(énoncé vide : عبارة فارغة)

مثال : **begin** a : = 1 ; b : = 2; **end**

يتبين من خلال العبارة المركبة الواردة في هذا المثال بأنها تحتوي على 3 عبارات ، إثنتين للتعيين وواحدة فارغة .

1.3 ـ تعيين ، تعبير (Affectation, expression)

إن العبارة الأكثر أساسية هي التعيين ، الذي يسمح بتخصيص متغيّر ، أو مركّب لمتغيّر ، بقيمة محسوبة حديثاً عن طريق تقييم لتعبير .



(énoncé d'affectation : عبارة تعيين ؛ expression : تعبير ؛ identificateur : معرّف)

إن قيمة التعبير ، من النوع T2 ، يجب أن تكون « متساوقة بالنسبة للتعيين » مع النوع T1 للمتغيّر المعيَّن ، هذا ما يتم ثبوته فور تحقق إحدى الخصائص التالية :

أ ـ T1 وT2 هما نفس النوع ولا يحتويان على النوع سجل .

ب ـ T1 من النوع حقيقي ، T2 من النوع صحيح (يوجد إذن تغيير أوتوماتي) . .
ج ـ T1 وT2 هما نوعان ترتيبيّان متساوقان (أنظر 7.3.2 (ب)) ، والقيمة من النوع T2 موجودة ضمن الفترة المحددة من قبل T1 .
(وكذلك
د ـ T1 وT2 هما نوعا مجموعتين متساوقان ، والمجموعة المعيَّنة يمكن أن تحوي القيمة .
هـ ـ T1 وT2 هما نوعا سلسلسال متساوقان) .
مثال :

```
type T=array [char] of boolean;
var a:integer;      b:real;      c:char;
     x:T;       y:'0'..'9';      y:array [boolean] of T;
```

a:= 3 تعيين سليم (أ)
b:= a تعيين سليم (ب)
c:= y تعيين سليم (ج)
x [chr (ord (y) + ord ('a') − ord ('0'))]:= true تعيين سليم
a:= b تعيين خاطىء (ب)
y:= '+' تعيين خاطىء (ج)
c:= a تعيين خاطىء (أ)
y [false]: = x تعيين صحيح (أ)

يمكن أن نعيّـن عدد صحيح لعدد حقيقي ، لكن العكس غير صحيح بتاتاً (إن هذا منطقي ذلك لأنه سيوجد في هذه الحالة فقدان للدقة (الأعداد العشريـة) غير متحكَّـم به) .

بما أن المتغير يفقد قيمته القديمة عند إجراء التعيين ، يجب إستعمال متغير مؤقت لإجراء عملية تبديل للقيم :

لتبديل قيم u وv نكتب :

t: = u; u: = v; v: = t

إن القيمة الأولية للمتغير هي غير محددة ؛ سيكون من الخطأ إستعمالها قبل أية عملية تعيين .

1.1.3 ـ تحليل

إن تعريفاً بسيطاً مثل u = v + 1 من التحليل ، يتطابق مع عبارة التعيين u: = v + 1 من البرنامج .

إن التعريف بالتكرار (التثنية إلى الوراء) للمتسلسلة u :

u = f (u, ...) وأوّل u = 0

تتم ترجمته إلى لغة الباسكال بـ

u: = 0 ; ... u: = f (u, ..)

مثال : لنفترض أننا نريد حساب الحدود الأولى من متسلسلة Fibonacci المعرّفة بـ

$$f_0 = 0, f_1 = 1 \text{ et } f_i = f_{i-1} + f_{i-2}$$
$$(f_2 = f_1 + f_0 = 1, f_3 = f_2 + f_1 = 2, f_4 = 3,...).$$

التعريفات		المعجم
نتيجة = أكتب ' = (', i, ')f' ,F لِـ i من 2 إلى 10	3	i - (صحيح)
أوّل F = 1	2	F - (صحيح) لـ i : f_i
أوّل G = 0	1	G - (صحيح) لـ i : f_{i-1}
$F = \overline{F} + \overline{G}$ $G = \overline{F}$		لِـ i ← f

لقد تم إدخال متسلسلة وسيطة G « لتخزين » القيم f_{i-2} المفقودة كلما تقدمت العمليات الحسابية .

بالفعل فإن كتابة f = f + f شيء لا يمكن ترجمته مباشرة إلى لغة الباسكال !

F هي إذن متسلسلة الـ f_i وG متسلسلة الـ f_{i-1} .

كذلك فإننا بحاجة إلى القيمة القديمة لـ F لكي نتمكن من تعريف G ، وإلى القيمة القديمة لـ G لتعريف F ؛ هذا ما يؤدي إلى إدخال معرّف مساعد t :

2	$F = \overline{F} + \overline{G}$
3	$G = t$
1	$T = \overline{F}$

```
program Fibonacci(output);
{ Fibonacci حساب الحدود الأولى من متسلسلة }
var i:integer;
    F,G,t:integer;   {F=f(i), G=f(i-1),
                      t متغير مساعد}
begin
  G:=0; F:=1;
  for i:=2 to 10 do begin
    t:=F; F:=F+G; G:=t;
    writeln('f(',i,')=',F)
  end
end.
```

ملاحظة : إن التفكير على مستوى « أكثر تجريداً » من عبارات برنامج تساعدنا هنا على الحصول وبسرعة على حلٍ سليم .

2.1.3 ـ تعبيرات (Expressions)

كونه مكوّناً من متأثرات (opérandes) (متغيرات ، ثوابت ، . . .) ومؤثرات (opérateurs) ، يسمح التعبير بتحديد قاعدة للحساب .

يجب أن يكون لكلٍ من المتأثرات قيمة (ترتيبية ، حقيقية ، أو مجموعة) ؛ إن الإستعمال في تعبير ، لمتغير ذي قيمة غير محددة يصبح غلطاً .

تترسّخ الأسبقيات الخاصة بالمؤثرات تبعاً للفئات الأربعة التالية والمعطيّة ضمن الترتيب للأسبقيات المتناقصة :

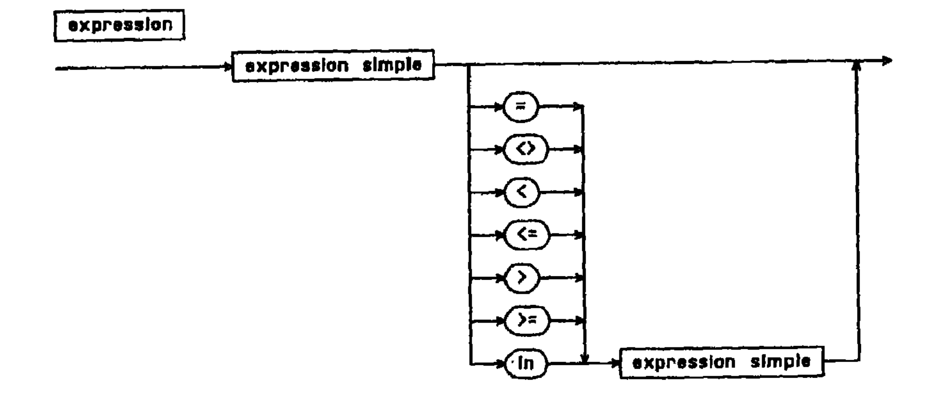
- المؤثر **not** (لا) ؛
- مؤثرات الضرب (**div, mod, * , / , and** (و)) ;
- مؤثرات الجمع (+ ، - ، (أو) **or**) والعلامات (+ ، -) ؛
- مؤثرات العلاقة (= ، < ، <= و <> ، >= ، > ، **in**) .

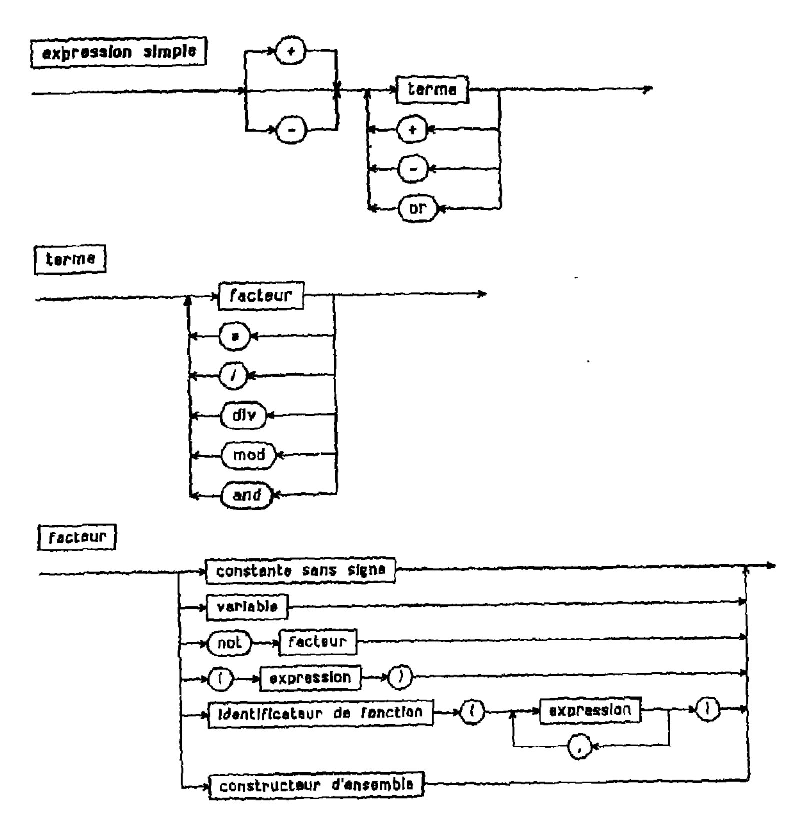
ضمن الأسبقية المتساوية ، يتم التقييم من اليسار إلى اليمين . يمكن دائماً تغيير ترتيب التقييم بواسطة الأقواس (( )) .

أمثلة :

```
a+b-c+d       تعني ──→  ((a+b)-c)+d
a+b*c-a/x     تعني ──→  (a+(b*c))-(a/x)
a/b/c         تعني ──→  (a/b)/c   soit   a/(b*c)
a/b*c         تعني ──→  (a/b)*c   soit   (a*c)/b
```

يجزّأ التعبير الى عوامل ، حدود وتعبيرات بسيطة على الشكل التالي :





(expression : تعبير ؛ simple : بسيط ؛ terme : حدّ ؛ facteur : عامل ؛ constante : ثابت ؛ sans signe : بلا علامة ؛ variable : متغير ؛ identificateur : معرّف ؛ fonction : دالّة ؛ constructeur : مشكّل ؛ ensemble : مجموعة).

أمثلة :

عوامل

```
x
15
(x+y+z)
sin (x+y)
not p
'HELLO'
```

حدود

```
x*y
i/(1-i)
(x=y) and (y<z)
k mod 2
```

تعبيرات بسيطة	`p or q` `x+y` `~x` `i*j+1`
تعبيرات	`x=1.5` `p=q` `p=q and r` `c in teinte`

إن نحو التعبير هو تكراري ؛ إنه يحتوي على عامِل الذي بدوره يحتوي على تعبير .

3.1.3 ـ مؤثرات حسابية :

الرمز	العملية	نوع المتأثرات	نوع النتيجة
–	طرح	صحيح وصحيح	صحيح
+	جمع	صحيح وحقيقي	حقيقي
*	ضرب	حقيقي وصحيح حقيقي وحقيقي	
/	قسمة	صحيح وصحيح صحيح وحقيقي حقيقي وصحيح حقيقي وحقيقي	حقيقي
div	قسمة صحيحة	صحيح وصحيح	صحيح
mod	معيار	صحيح	صحيح

ملاحظة : كذلك نستعمل + ، – ، * كمؤثرات على المجموعات .

x / y هي دائماً من النوع حقيقي وتكون غلط إذا كانت 0.0 = y

x div y هي غلط إذا 0 = y

i mod j هو غلط إذا $j \leqslant 0$

في حال كانت $i \geqslant 0$ و $j < 0$ يصبح :

$$i = (i \text{ div } j) * j + i \text{ mod } j$$

إن حاصل قسمة i بـ j هو **i mod j**

تُطبَّق المؤثرات ذوات المتأثر الواحد + (تطابق) و – (إنقلاب العلامة) على الأعداد الصحيحة والحقيقية ؛ تكون النتيجة من نوع المتأثر .

يطلق إسم **التعبير المختلط** على التعبير الذي تتم فيه تغييرات أوتوماتية من النوع صحيح إلى النوع حقيقي : *1 + 2.0/4 (1 → 1.0 , 4 → 4.0)*

4.1.3 ـ مؤثرات بولية

الرمز	العملية	نوع المتأثرات	نوع النتيجة
(و)	عطف	بولي وبولي	بولي
(أو)	تفرُّق		
(لا)	نفي	بولي	

p and q

q \ p	خطأ false	صح true
false	*false*	*false*
true	*false*	*true*

p or q

q \ p	*false*	*true*
false	*false*	*true*
true	*true*	*true*

p	**not** *p*
false	*true*
true	*false*

إنها المْؤثرات التقليدية المستعملة في جَبْر بُول . هناك مؤثرات أخرى تتفرع عنها (أو المقتصرة (ou exclusif) ، لا ـ و ، . . .) .

5.1.3 ـ مؤثرات العلاقة

الرمز	العملية	نوع المتأثرات	نوع النتيجة
=	مُساوٍ لـ	كل نوع بسيط أو كل سلسال	بولي
< >	مُختلفٌ عن		
<	أقل من		
>	أكبر من		
< =	أقل أو مساوٍ لـ		
> =	أكبرُ أو مساوٍ لـ		

على المتأثرات أن تكون من أنواع متساوقة (أنظر 7.3.2) <= ، >= ، = ،
<> وin هم كذلك مؤثرات على المجموعات (أنظر 1.3) ؛ = و<> تطبّق على الأنواع دليل (Pointeur) (أنظر 6.3) .

6.1.3 ـ دوال معرّفة مسبقاً

إلى هذه المؤثرات ، تضاف مؤثرات مقدّمة على شكل دوال ؛ تكون هذه الدوال حسابية ، ترتيبية ، للنقل ، أو بولية .

دوال حسابية (Fonctions arithmétique)

على متأثر x من النوع على السواء حقيقي أو صحيح ، تعطي هذه الدوال نتيجة « حقيقي » ، ما عدا abs وsqr حيث تكون للنتيجة نوع المتأثر x .

abs (x)	تحسب القيمة المطلقة (حقيقي أو صحيح) لـ x
sqr (x)	تحسب مُربّـع (حقيقي أو صحيح) x
sin (x)	تحسب جيب الزاوية x (x بالراديان)
cos (x)	تحسب جيْب التمام للزاوية x (x بالراديان)
arc tan (x)	تحسب القيمة بالراديان لقوس ظل x
exp (x)	تحسب e^x
ln (x)	تحسب اللوغاريتم الأعلى لـ x ، في حال $x > 0$
sqrt (x)	تحسب الجذر التربيعي غير السالب لـ x ، في حال $x > 0$

(يكتب كذلك الرفع إلى قوة x^y : exp (y * ln (x))

دوال ترتيبيّـة (fonctions ordinales)

succ (x) : x هو تعبير من نوع ترتيبي ، والنتيجة من نفس النوع ؛ إنها القيمة التي يكون عددها الترتيبي مباشرة أكبر من ذلك الخاص بـ x (إذ وُجِدْ)

pred (x) : x هو تعبير من نوع ترتيبي ، والنتيجة من نفس النوع ؛ إنها القيمة التي يكون عددها الترتيبي مباشرة أقل من ذلك الخاص بـ x (إذا وُجِدْ)

ord (x) : تعطي العدد الترتيبي ، من نوع صحيح ، للمتأثر x ؛ x هو تعبير من نوع ترتيبي .

chr (x) : تعمل على المطابقة للتعبير x ، من النوع صحيح ، السمة التي يكون عددها الترتيبي هو قيمة x (إذا وُجِدْ) . إن العلاقة chr (ord (c)) = c هي صحيحة بالنسبة لكل سمة c .

دوال النقل (Fonctions de transfert)

trunc (x) : كَوْن x من النوع حقيقي ، فإننا نحصل على « القسم الصحيح » :

القسم الصحيح لِـ x إذا $x \geqslant 0$

ـ القسم الصحيح لِـ abs (x) إذا $x < 0$

أمثلة : trunc (17.986) = 17 ،

trunc (− 4.3) = − 4

round (x) : كَوْن x من النوع حقيقي ، فإننا نحصل على « العدد الصحيح الأقرب » :

trunc (x + 0.5) إذا $x \geqslant 0$

trunc (x − 0.5) إذا $x < 0$

أمثلة : round (− 4.3) = − 4   round (17.986) = 18

round (− 3.5) = − 4   round (3.5) = 4

مثال : **var** i : integer ; r: real

r : = 9.6 ; i: = (trunc (r) + round (r)) **mod** 2

i تساوي 1

دوال بوليّـة (Fonctions booléennes)

odd (x) تساوي true (صحّ) إذا كان التعبير x من النوع الصحيح هو مفرداً ، وإلاّ فإنها تساوي false (خطأ) (إن odd (x) هي abs (x) **mod2** = 1) .

تطبّـق الدوال eof وeoln على السجلات والنصوص :

eof (f) تعني بأن النافذة قد وصلت إلى نهاية السجل f

eoln (f) تعني بأن النافذة قد وصلت إلى نهاية سطر في النص f

إذا أُلغيَ الوسيط f فهذا يعني بأن المقصود هو السجل input

مثال : إحسب الخطوط الخاصة بحساب المثلثات والجذر التربيعي لقيمة معطيّـة .

البرنامج

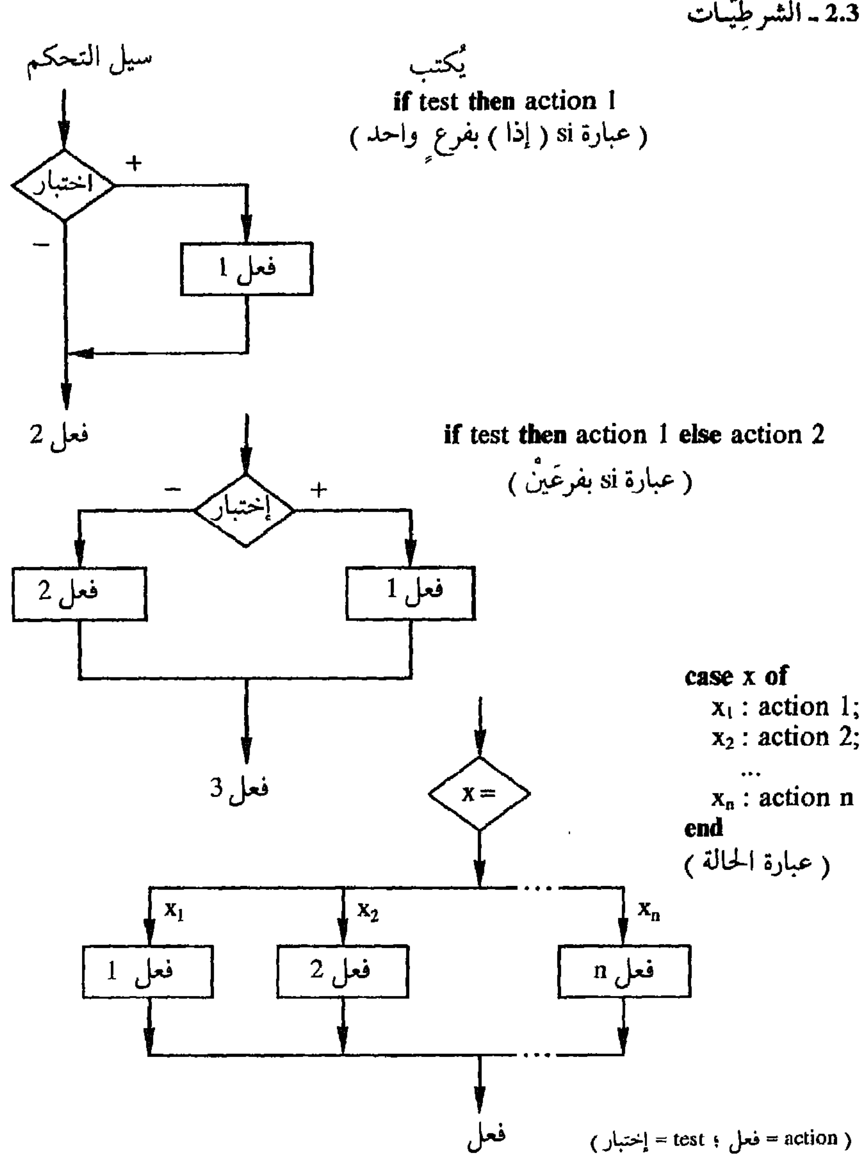
```
program lignes(input,output):
{ خطوط المثلثات والجذر التربيعي }
var x:real;        { قيمة معطيّـة بالراديان }
begin
  read(x);
  writeln('x':12, 'sin':12, 'cos':12);
  writeln(x:12:5, sin(x):12:5, cos(x):12:5);
  writeln;
  writeln('tg':12,'cotg':12,'racine':12);
  writeln(sin(x)/cos(x):12:5, cos(x)/sin(x):12:5,
          sqrt(x):12:5)
end.
```

يعطي هذا البرنامج النتيجة :

```
       x          sin          cos
 0.10000      0.09983      0.99500

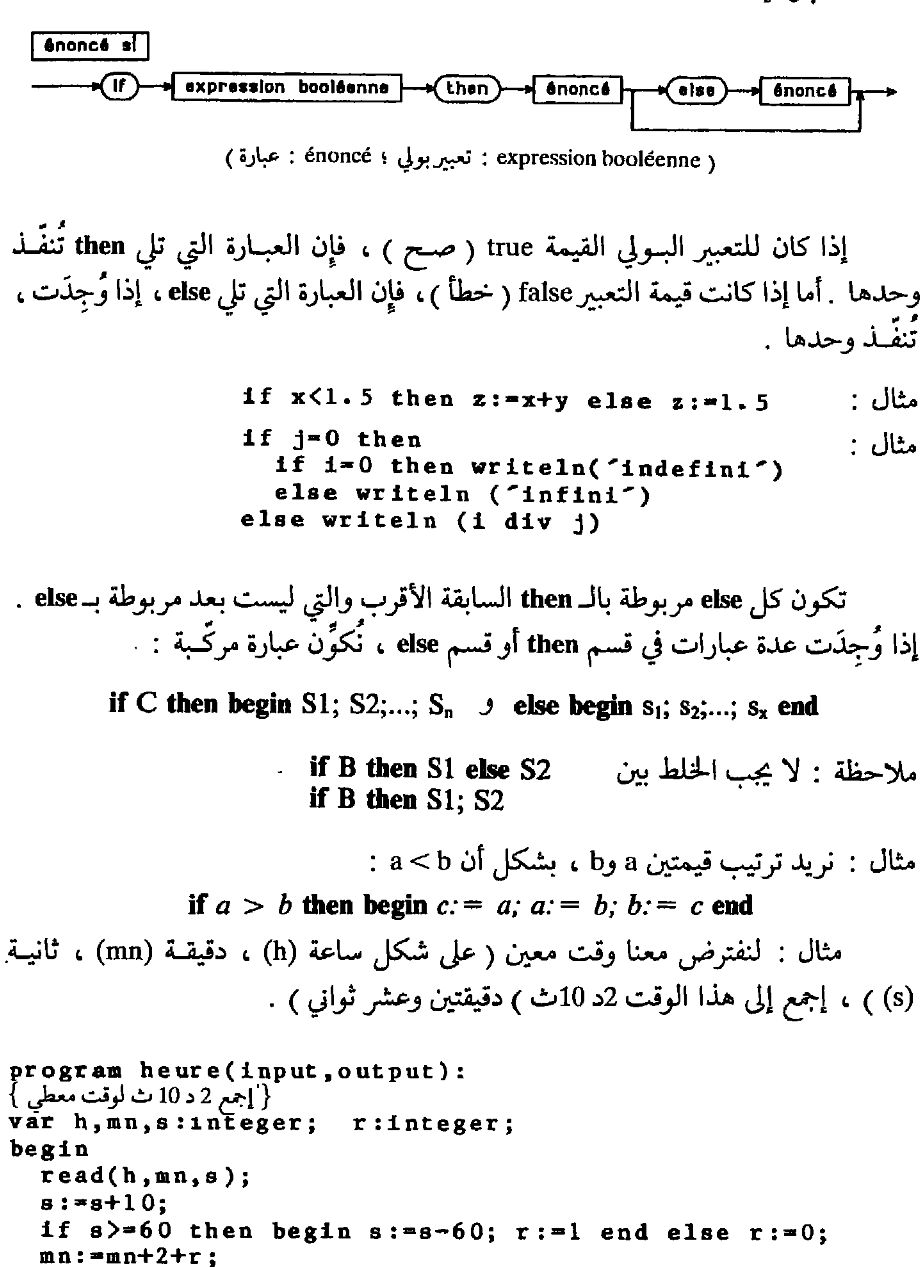
      tg         cotg       racine
 0.10033      9.96664      0.31623
```

2.3 ـ الشَرطِيّات



(action = فعل ؛ test = إختبار)

1.2.3 ـ عبارة إذا (Enoncé si)



(expression booléenne : تعبير بولي ؛ énoncé : عبارة)

إذا كان للتعبير البـولي القيمة true (صـح) ، فإن العبـارة التي تلي then تُنفّـذ وحدها . أما إذا كانت قيمة التعبير false (خطأ) ، فإن العبارة التي تلي else ، إذا وُجِدَت ، تُنفّـذ وحدها .

مثال :

```
if x<1.5 then z:=x+y else z:=1.5
```

مثال :

```
if j=0 then
  if i=0 then writeln('indefini')
  else writeln ('infini')
else writeln (i div j)
```

تكون كل else مربوطة بالـ then السابقة الأقرب والتي ليست بعد مربوطة بـ else .
إذا وُجِدَت عدة عبارات في قسم then أو قسم else ، نُكوِّن عبارة مركّـبة :

if C **then begin** S1; S2;...; S_n و **else begin** s_1; s_2;...; s_x **end**

ملاحظة : لا يجب الخلط بين **if** B **then** S1 **else** S2
if B **then** S1; S2

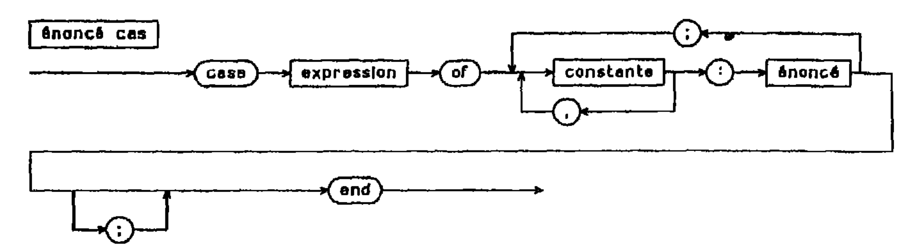
مثال : نريد ترتيب قيمتين a و b ، بشكل أن a < b :

if $a > b$ **then begin** $c := a;\ a := b;\ b := c$ **end**

مثال : لنفترض معنا وقت معين (على شكل ساعة (h) ، دقيقـة (mn) ، ثانيـة (s)) ، إجمع إلى هذا الوقت 2د 10ث (دقيقتين وعشر ثواني) .

```
program heure(input,output):
{'إجمع 2 د 10 ث لوقت معطي }
var h,mn,s:integer;  r:integer;
begin
  read(h,mn,s);
  s:=s+10;
  if s>=60 then begin s:=s-60; r:=1 end else r:=0;
  mn:=mn+2+r;
  if mn>=60 then begin mn:=mn-60; r:=1 end else r:=0;
  h:=(h+r) mod 24;
  writeln (h,mn,s)
end.
```

2.2.3 ـ عبارة الحالة (Enoncé cas)



(ثابت : constante ؛ تعبير : expression ؛ عبارة : énoncé)

يتم تقييم التعبير : قيمته تؤدي إلى تنفيذ العبارة الموافقة لثابت الحالة الذي يعبِّـر عن هذه القيمة . يجب أن تكون كل ثوابت الحالة مختلفة ومن نفس النـوع الترتيبي الـذي للتعبير . إذا لم يكن أي من ثوابت الحالة مساوياً لقيمة التعبير ، يُعَدّ هذا غلطاً .

إنّ « ؛ » قبل الـ **end** هو إختياري .

مثال :

```
case caractereLu of
   '+' : x:=a+b;
   '-' : x:=a-b;
   '*' : x:=a*b;
end
```

(السمة المقروءة : caractere lu)

أو كذلك مع التصريح **var** *opérateur : (plus, moins, fois)*

```
case operateur of
  plus : x:=x+y;
  moins: x:=x-y;
  fois : x:=x*y;
end
```

(* : fois ؛ - : moin ؛ + : plus ؛ مؤثر : opérateur)

التعبير

```
if a<b then c:=a else c:=b
```

يُكْتَب كذلك :

```
case a<b of
false: c:=b;
true : c:=a
end
```

مثال : إطبع العمل الذي يجب القيام به في كل يوم من الأسبوع :

```
program semaine(input,output);
type jour=(lundi,mardi,mercredi,jeudi,vendredi,samedi,
           dimanche);
var j:jour;    k,n:integer;    {j=ord(jour)}
begin
  read(n);
  j:=lundi; for k=1 to n do j:=succ(j);
  case j of
    lundi,mardi,mercredi,jeudi: writeln('au travail');
    vendredi: writeln('au travail, pour 7 heures');
    samedi: writeln('on sort ce soir');
    dimanche: writeln('on recupere')
  end
end.
```

(semaine : أسبوع ؛ lundi : الإثنين ؛ mardi : الثلاثاء ؛ mercredi : الأربعاء ؛ jeudi : الخميس ؛ vendredi : الجمعة ؛ samedi : السبت ؛ dimanche : الأحد ؛ au travail : إلى العمل ؛ pour 7 heures : لسبع ساعات ؛ on sort ce soir : نخرج هذا المساء ؛ on recupere : نستردّ قوانا) .

ملاحظة : عندما يأخذ التعبير الذي يلي case ، قيمة خارجة عن قائمة ثوابت الحالة ، فهذا غلط . في بعض الآلات ، نستعمل تمديداً (خارج إطار القاعدة AFNOR) للغة الباسكال : عبارة مُدْخلة بـ else أو otherwise تجمَعُ كل الحالات غير المُعَالَجة صَراحة :

```
case op of
  '+':u:=u+v;
  '-':u:=u-v;
  else:writeln('operateur inconnu')
end
```

(opérateur inconnu : مؤثر مجهول)

وإلاّ نتجنب المشكلة عن طريق التحقُّق المُسبق لقيمة التعبير :

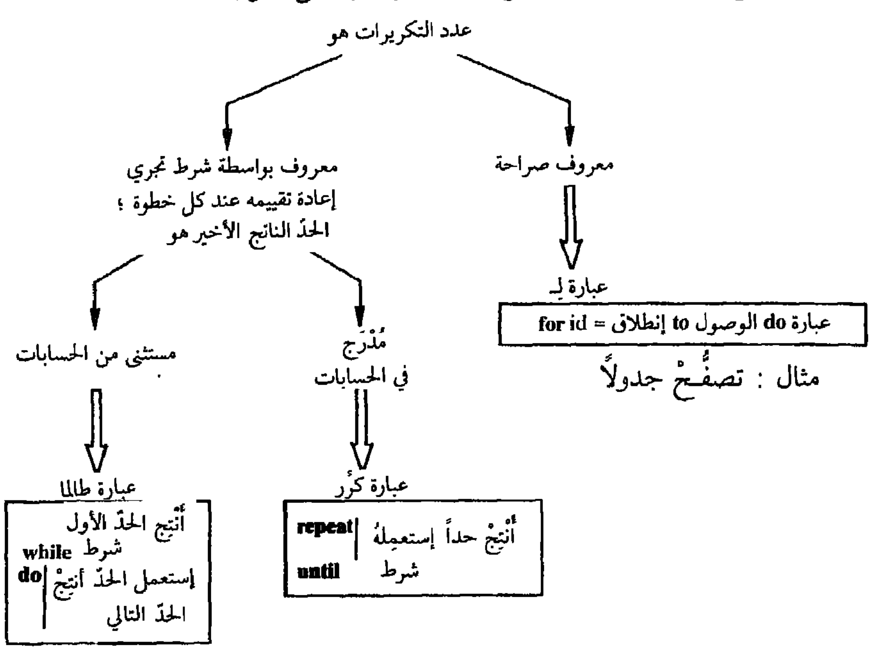
```
if (op='+') or (op='-') then
  case op of
    '+':u:=u+v;
    '-':u:=u-v
  end
else
  writeln('operateur inconnu')
```

أو بطريقة أشمل :

if *op* **in** *['+', '−']* **then** ... Cf. 4.4).

3.3 ـ طريقة تكراريّـة (Itéfation)

تعرض لغة الباسكال ثلاثة طرق لتنفيذ عبارة بشكل مكرّر :

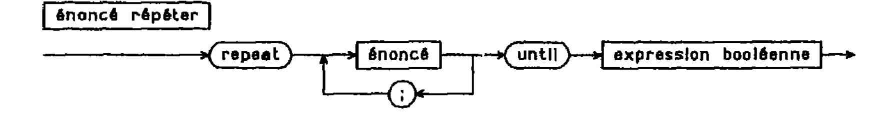


الإقتِصـار (Exclusion) : « توقيف معلومـاتي » ، لا تنفع المعـطِيّـة الأخيـرة إلاّ للإشارة الى نهاية الحسابات .

التضـمين (Inclusion) : البحث عن الحدّ الأول لمتسلسلة والـذي يُحقِّق خاصـة معيّـنة .

لا تغطّـي هذه الكتابات الثلاثة كل الحاجات . يجب وبوجـه خاص أن تتمكّـن التكرارية من التوقف فور ظهور غلط أو ظهور عدم ترابط في المعطيات : إن عبارة الطفور (branchement) go to تسمح بهذا التوقف المفاجيء .

1.3.3 ـ عبارة كرِّر (Enoncé répéter)



(expression booléenne : تعبير بولي ؛ énoncé : عبارة) .

يتم تنفيذ متسلسلة العبارات بطريقة مكرّرة حتى يأخذ التعبير البولي القيمة true (صح) يتم تنفيذ متسلسلة العبارات مرة واحدة على الأقل ، ذلك لأنه يتم إختبار التوقف بعد التنفيذ .

إنتبه : يجب أن تستطيع متسلسلة العبارات تغيير قيمة التعبير البولي (وإلاّ فإن الحلقة لا تنتهي) :

repeat V = f (V) ; cond = g (V) **until** cond

(cond (إختصار لـ condition) = شرط)

مثال : numdec (عدد عشري)

لنفرض أننا نريد تحويل قيمة رقميّة V صحيحة موجبة وأقل من 99999 إلى تمثيلها العشري (على شكل سمات) .

إن للقيمة V التمثيل العشري «$t_4\, t_3\, t_2\, t_1\, t_0$» حيث يكون t_0 رقم الآحاد ، t_1 رقم العشرات ، t_i الرقم المعيَّـن بالمُعامِل 10^i . إن القيمة التي يمثلها t_i (لنفرض (– ord (t_i) ord ('0')) هي إذن :

(v div 10^i) mod 10 : t_0 = *chr (ord ('0') + v* **mod** *10)*
t_1 = *chr (ord ('0') + (v* **div** *10)* **mod** *10)*
...
t_4 = *chr (ord ('0') + (v* **div** *10000)* **mod** *10)*

هذا ما يؤدي إلى تعـريف متسلسلتين تكـراريتين (مثنى إلى الـوراء) T (Suites récurrentes) و R .

$$\begin{cases} T_i = R_{i-1} \textbf{ mod } 10 & \text{مُتسلسلة قيم الأرقام العشرية} \\ R_i = R_{i-1} \textbf{ div } 10 & \text{متسلسلة « الحواصل »} \\ R_{-1} = v \end{cases}$$

مع t_i = chr (ord ('0' + T_i)

نَحسُب هكذا الأرقام المتتالية (t_0 إلى t_4) بالترتيب المعاكس لإستعمالهم الطبيعي : نستعمل جدولاً لتخزينهم . يبدأ الحساب مع $R_1 = V$ وينتهي عند n بشكل أن $R_{n-1} = 0$ أي t_K = '0' بالنسبة لِـ $K \geqslant n$.

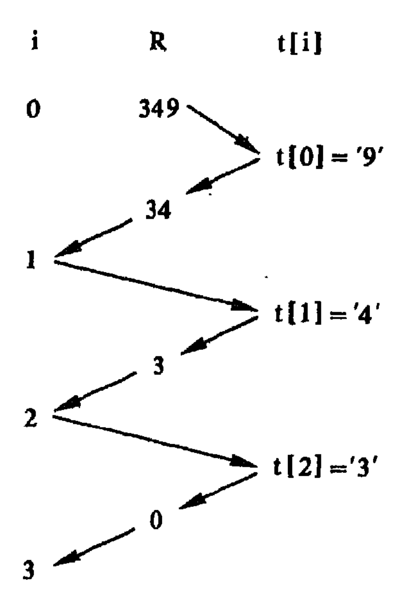
```
program numdec(input,output);
{ تحويل قيمة رقمية صحيحة إلى تمثيلها العشري على شكلِ سمات }

var v:integer;              { القيمة المطلوب تحويلها }
    t:array[0..4] of char; { التمثيل العشرى
                             { آحاد [ 4 ] t ، عشرات [ 3 ] t ، . . .
```

```
    R:integer;          { متسلسلة الحواصل المطلوب تحويله }
    i:integer;          { دليل الرقم المحسوب ، في t }
    j:integer;          { لتصفح الجدول }
begin
  read(v); R:=v; i:=0;
  repeat
    t[i]:=chr(ord('0')+R mod 10);
    R:=R div 10;
    i:=i+1
  until R=0;
  for j:=i-1 downto 0 do write(t[j]); writeln
end.
```

نستعمل عبارة كرِّر لأن آخر t [i] محسوب هو رقم غير الصفر ، فإذن مُعبِّر ؛ وكذلك لأنه يوجد على الأقل رقم يجب حِسابُهُ t [0] ، فإذن على الأقل تنفيذ واحد للحلقة .

خلال عملية الحساب ، عندنا مثلاً بالنسبة لِـ V = 349



لتحويل قيمة رقمية الى تمثيلها الثنائي (binaire) (في القاعدة 2) أو الثماني (octale) (في القاعدة 8) ، يكفي أن نستبدل في التكرارية الثابت 10 بـ 2 أو بـ 8 .
للتحويل الى التمثيل السادس عشري (hexadécimale) (في القاعدة 16) ، نستبدل 10 بِـ 16 وتَتِمّ عملية تحويل قيمة الرقم (0 إلى 15) إلى سمة (,'9' ,'A' ,'B' ,'C' ,'D' ,'E' ,'F' '0' ,'1' , ... ,'8') عن طريق التقسيم (indexation) للجدول C الحاوي للسمات

(*T* : **array** *[0..15]* **of** *char* et *T [0]: = '0', T [1]: = '4', ... , T [10]: = 'A', ... T [15] = 'F')* :

repeat *t [i]: = C [R* **mod** *16]*

2.3.3 ـ عبارة طالما (Enoncé tant que)



تنفّـذ العبارة بطريقة مكررة طالما يأخذ التعبير البولي القيمة true (صح)
لا تنفذ العبارة بتاتاً إذا كان للتعبير البولي القيمة false (خطأ) عند الإنطلاق .

إنتبه : يجب أن تستطيع العبارة تغيير قيمة التعبير البولي (وإلاّ فإن الحلقة لا تنتهي)

while cond **do** $\begin{cases} V = f(\bar{V}) \\ cond = g(V) \end{cases}$

(cond (إختصار لـ condition) = شرط)

إذا كان يجب ظهور عدة عبارات في قسم **do** ، نُكوِّن منها عبارة مركّـبة :

while c **do begin** énoncé; ... énoncé **end**

العبارة while b do E هي مكافئة لِـ :

begin if b **then repeat** E

until not (b)

end

لا يجب أن ننسى إنتاج الحدّ الأول قبل بدء تنفيذ عبارة while :

$V = V_0$; **while** cond (V) **do** $V = f(\bar{V})$

مثال : binnum (عدد بالتمثيل الثنائي)

لنفرض أننا نريد التحويل إلى قيمته الرقمية ، عدد موجب صحيح معطي بالتمثيل الثنائي (متسلسلة من السمات ، ممثّلة القيمة في القاعدة 2) .

إن للقيمة V التمثيل الثنائي «$t_n \ldots t_4 t_3 t_2 t_1 t_0$» حيث أن t_i مُلحق بالمُعامل 2^i . إذا سمّينا T_i القيمة المطابقة للسمة t_i ،(T_i = ord (t_i) – ord ('0'))، نحصل على :

$$v = T_n2^n + \ldots + T_32^3 + T_22^2 + T_12^1 + T_02^0$$
$$= (((\ldots(T_n2 + T_{n-1})2 + T_{n-2} \ldots + t_3)2 + t_2)2 + t_1)2 + t_0$$

(إن هذا هو إنبساط هورنر (développement de Horner) لمتعدد الجذور) .

إذن تعرّف القيمة V بواسطة متسلسلة تكراريّة :

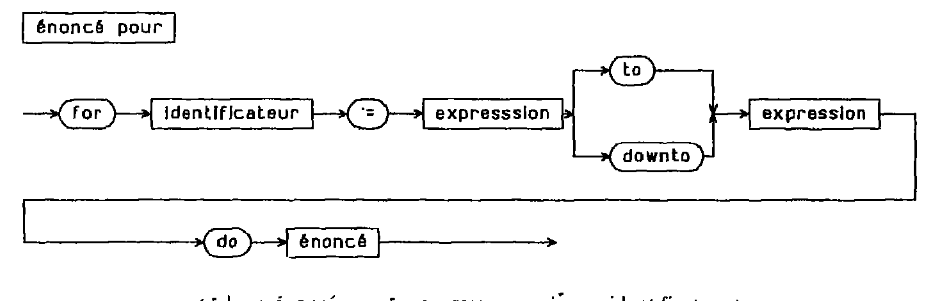
$$v = v_0 \qquad \begin{cases} v_i = v_{i+1} \star 2 + \bar{T}_i \\ v_{n+1} = 0 \end{cases}$$

```
program binnum(input,output);
{ التحويل إلى قيمة رقمية لقيمة ثنائية }
var t:char;        { رقم ثنائي مُعتاد }
    v:integer;     { قيمة }
begin
  v:=0;
  read(t);                                    ← إنتاج العنصر الأول
  while (t='0') or (t='1') do begin
    v:=v*2+ord(t)-ord('0');                   ← إستعمال العنصر الأخير المُنتج
    read(t)
  end;
  writeln (v)                                 ← إنتاج العنصر التالي
end.
```

نستعمل عبارة while لأن آخر t مُنتج ليس رقماً من العدد .

3.3.3 ـ عبارة لِـ (Enoncé pour)

لا تُستعمل العبارة « لِـ » إذا كان عدد التكريرات معروفاً . إن فائدتها تكمن في أنها تقدم « عدّاداً » ، أمر كثير المنفعة مثلاً لتصفّح جدولاً .



(identificateur : معرّف ؛ expression : تعبير ؛ énoncé : عبارة)

إن المتغيّـر والمسمى متغير التحكّـم ، يأخذ بالتتابع كل القيم من تعبير الإنطلاق حتى تعبير الوصول ؛ لكل قيمة ، يتم تنفيذ العبارة .

ـ مع **to** ، يتم الإنتقال الى القيمة التالية بواسطة succ

ـ مع **downto** ، يتم ذلك الإنتقال بواسطة pred

ـ يكون متغير التحكـم من **نوع ترتيبي** ؛ يجب أن تكون قيم الإنطلاق والوصول من نوع متساوق مع هذا النوع .

ـ يجب أن يكون متغير التحكـم **بسيطاً** (معرَّف ، غير مركَّـب) وموضعياً (مصرَّح في قسم تصريحات متغيرات الفدرة الفعّـالة) .

ـ بعد إنتهاء العملية التكرارية ، تكون قيمة متغير التحكُـم **غير محدّدة** .

ـ لا يجب بتاتاً التغيير المباشر لمتغير التحكم .

ـ إذا كانت قيمة الإنطلاق أكبر من قيمة الوصول (مع **to** ، أو أقل مع **downto**) ، فإن العبارة لا تنفّـذ .

إذا وضعت هذه القيود على حِدَة ، فإن العبارة :

for v : = e1 to e2 do E

مكافئة لِـ :

```
begin t1:=e1;  t2:=e2;
  if t1<=t2 then begin
    v:=t1; E;
    while v<>t2 do begin
      v:=succ(v); E end
  end
end
```

حيث تكون t1 وt2 متغيرات مساعدة لا تظهر في مكان آخر من البرنامج .

مثال : Trigo (الخطوط الخاصة بحساب المثلثات)

لنفرض أننا نريد طبع جدولاً للخطوط الخاصة بحساب المثلثات ، جيب (sinus) وجيب التمام (cosinus) على الشكل :

DEGRES	SIN	COS	
0	0.000	1.000	90
1	0.017	1.000	89
2	0.035	0.999	88
3	0.052	0.999	87
.	.	.	.
.	.	.	.
.	.	.	.
21	0.358	0.934	69
22	0.375	0.927	68
	COS	SIN	DEGRES

```
program trigo(output);
{table sinus/cosinus, en degrés}
var D:integer; {angle en degrés}
    R:real;    {angle en radians}
    i:integer;
begin
    {titre haut}
  for i:=1 to 30 do write ('-'); writeln;
  writeln('I DEGRES I','SIN':7,'I':3,'COS':7,'I':3);
  for i:=1 to 39 do write('-'); writeln;
    {table}
  for D:=0 to 22 do begin
    R:=3.14159265*D/180.0;
    writeln('I',D:5,'I':4,sin(R):7:3,'I':3,
             cos(R):7:3,'I':3,90-D:5,'I':4)
  end;
    {titre bas}
  for i:=1 to 39 do write('-'); writeln;
  writeln('I':10,'COS':7,'I':3,'SIN':7,'I':3,
          ' DEGRES I');
  write(' ':10); for i:=10 to 39 do write ('-'); writeln
end.
```

(degré : درجة ؛ angle : زاوية ؛ titre haut : العنوان العلوي ؛ titre bas : العنوان السفلي)

مثال : فرز بمبادلات متتالية .

إن فرز متسلسلة من n عنصر بالترتيب التزايدي يقوم على ترتيبها بشكل أن كل عنصر يكون أصغر ، أو يساوي ، من كل العناصر التي تليه : $j < i \Leftarrow T_i \geqslant t_i$.

هناك فكرة سهلة تقوم على العمل بالتكرار (أو التثنية إلى الوراء) .

- إذا وضعنا في الموقع الأوّل من المتسلسلة ذات الـ i عنصر ، العنصر الأصغر ، نكون أمام فرز من $i - 1$ عنصر ؛
- إن متسلسلة من عنصر واحد تُعدُّ مفروزة .

```
program tri(input,output);
{ فرز بالمبادلات المتتالية }
const n=5;              { عدد العناصر المطلوب فرزها }
var T:array [1..n] of integer;   { عناصر للفرز }
    i,j,k,L:1..n;
    aux:integer;
begin

  { إقرأ المتسلسلة الأولية }
  for k:=1 to n do read(T[k]);

  { الحلقة i : ضع في T [ i ] الأدنى من T [ i...n ] }
```

```
  for i:=1 to n-1 do
     { الحلقة j : قارن T [ i ] بكل T [ i + 1...n ] }
    for j:=i+1 to n do
      if T[i]>T[j] then begin      { إذن بادِل }
        aux:=T[i]; T[i]:=T[j]; T[j]:=aux end;

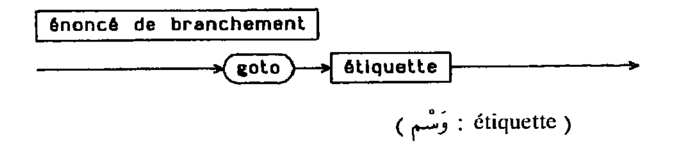
  { اكتب المتسلسلة المرتّبة }
  for L:=1 to n do write(' ',T[L]); writeln
end.
```

مثال :

	4	9	2	0	3
i = 1	2	9	4	0	3
	0	9	4	2	3
i = 2		4	9	2	3
		2	9	4	3
i = 3			4	9	3
			3	9	4
i = 4				4	9
إذن	0	2	3	4	9

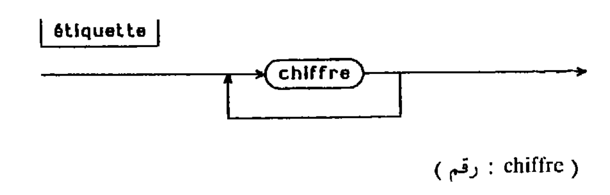
إن مدّة المعالجة (لعدد كبير n) متناسبة مع عدد المقارنات ؛ هذا الفرز هو « فرز بـ n^2 » إنها إحدى الطرق الأسهل للبرمجة ، لكنها إحدى الأبطأ في التنفيذ .

4.3.3 ـ عبارة الطفور (التفريغ) (Enoncé de branchement)



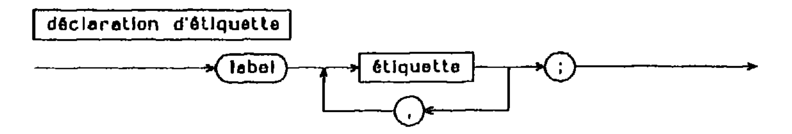
(وَسْم : étiquette)

تعني عبارة الطفور أن التنفيذ يجب أن يُتَابع من النقـطة المشار إليهـا بالـوسم في البرنامج . (إنها يمكن أن تُسبِّب الإنهاء الإجباري لعدة إجراءات أو دوال منشَّـطة) .

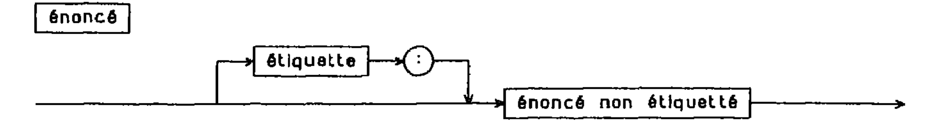


(رقم : chiffre)

يتم تمييز الوسـومات بـواسطة قيمتهم ، في الفتـرة 0....9999 (4 أرقام) . يجب التصريح عن كل وسم :



وكذلك تعريفه (بطريقة واحدة) :



(énoncé non étiquetté : عبارة غير موسومة) .

مثال : تردُّد (fréquence)

أحصِ ظهور الأحرف في نصٍ لا يحتوي إلّا على أحرف كبيرة وتباعدات . في حال ظهور سمة أخرى ، يعدّ ذلك غلطاً .

ملاحظة : في كل لُعب سمات ، تكون الأحرف من 'A' إلى 'Z' مرتّبة لكنِ ليس بالضرورة متتالية ((succ ('A') ≤ 'B' ؛ لا يعمل هذا البرنامج بشكل كامل إلّا عـلى حـاسب آلي تكـون الأحـرف فيـه مكـوّدة بـطريقـة متـاليـة (لعب سمـات ASCII ، DISPLAY ، BCD . . .) .

```
program frequence(input,output);
{ أحص ظهور الأحرف فى نص يحتوي فقط على أحرف كبيرة وتباعدات .
في لُعب السمات يجب أن تكون الأحرف متتالية }
label 1;    { للتوقف في حال وجود غلط في المعطيات }
var F:array ['A'..'Z'] of integer;         { تردّد }
    i,j:'A'..'Z';        c:char;   { سمة مقروءة }
begin
  for i:='A' to 'Z' do F[i]:=0;
  while not eof do begin
    read(c);
    if (c<>' ') and ((c<'A') or (c>'Z')) then begin
      writeln('erreur - caractere lu: <',c,'>');
      goto 1
    end;
    F[c]:=F[c]+1
    end;
  for j:='A' to 'Z' do writeln(j,F[j]:6);
1:end.
```

(erreur : غلط ؛ caractere lu : سمة مقروءة)

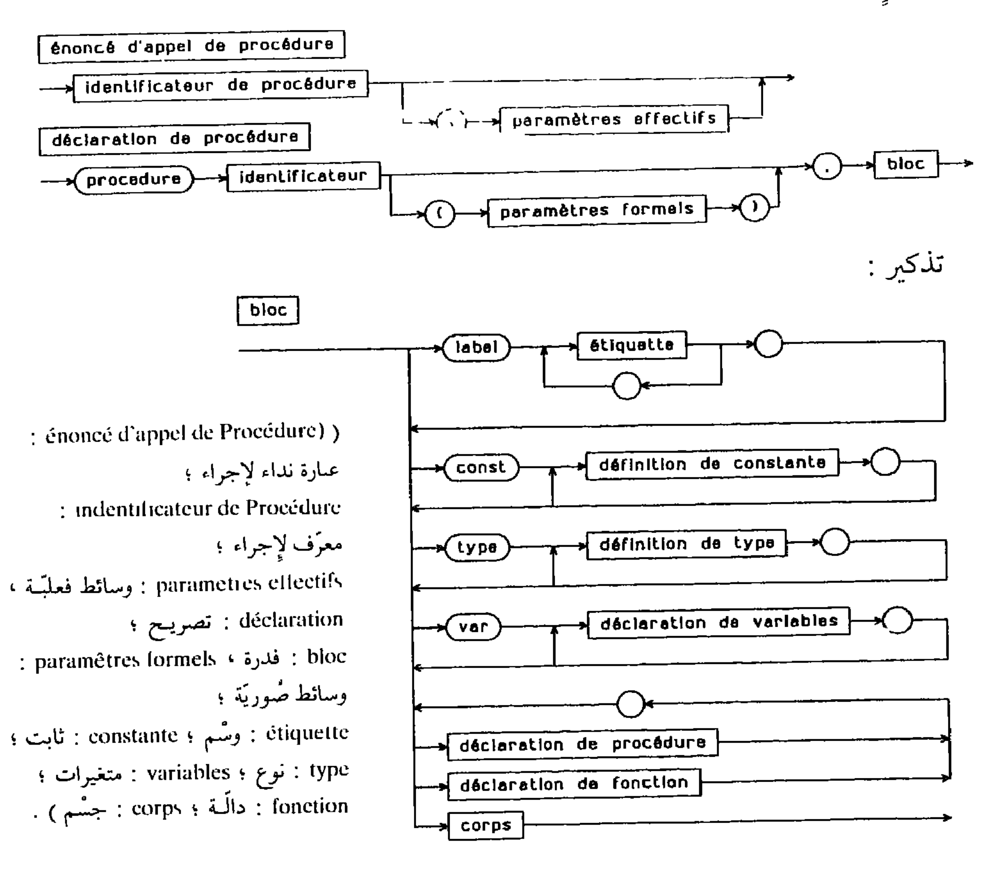
إنتبه : إن استعمال عبارات الطفور تجعل وبسرعة أيّ برنامـج ، صعب القراءة وتُضِرُّ بحُسنِ سير عمله سواء كان ذلك في لغة الباسكال أو في أية لغة أخرى .

4.3 - إجراء (Procédure)

يمكن إعطاء إسماً (معرِّف) لعبارة ، والرجوع إليها بواسطة هذا الإسم ؛ تصبح العبارة إجراءا ، التصريح عنها **تصريحا لإجراء** ، ونرجع إليه بواسطة عبارة **نداء لإجراء** .

يمكن لتصريح كهذا أن يحتوي على تصريحات وتعريفات (لمتغيرات ، لأنـواع ، . . .) ، وكذلك على تصريحات أخرى لإجراءات ؛ هكذا فإن الأدوات المُعرَّفة لا يمكن إستعمالها إلاّ في داخل الإجراء نفسه وتسمى **موضعيّـة** في الإجراء . ليس لمعرِّفيهم أي معنى إلاّ في الفدرة التي يشكلها تصريح الإجراء .

غالباً ما نستعمل إجراءات ، إمّا لتسمية عبارة تُستعمل مرات عديدة ، إمّـا لتحسين قراءة البرنامج عن طريق تحويل أقسام ثانوية من التحليل إلى إجراءات . يتعلّـق الإجراء بفعلٍ محدّد ومنفرد .



تذكير :

(énoncé d'appel de Procédure : عبارة نداء لإجراء ؛ indentificateur de Procédure : معرّف لإجراء ؛ paramètres effectifs : وسائط فعليّـة ، déclaration : تصريـح ؛ bloc : فدرة ، paramètres formels : وسائط صُوريّة ؛ étiquette : وسْم ؛ constante : ثابت ؛ type : نوع ؛ variables : متغيرات ؛ fonction : دالّـة ؛ corps : جسْم) .

1.4.3 ـ كَشْف

نبحث في حسابات بنك عن الأرصدة المدينة غير المغطَّاة بتسهيلات مصرفية مسموح بها (مأذونة) .

تكون المعطيات على الشكل :

رقم الحساب	وَضْعُهُ	الإِذْنُ
(4 أرقام)	(حقيقي)	(حقيقي)

تُعالج فقط الأرصدة التي تنتهي أرقام حساباتها بصفر ؛ تنتهي المعطيات بحساب رقمه 0000 والذي هو خارج إطار المعالجة .

تكون العملية التكرارية إذن من النوع توقُّف مع إقْتِصار والذي يُتَرجَم بمتتالية :

```
lire les donnees du premier compte
while compte < > dernier do begin
   traiter le compte, en comparant position
      et autorisation
   lire les données du prochain
      compte à traiter
end
```

```
إقرأ معطيات أوّل حساب
while الحساب < > آخر do begin
   عالج الحساب بمقارنة
      الوضع والإذن
   إقرأ معطيات الحساب التالي
      المطلوب معالجته
end
```

إن الفعل « إقرأ معطيات حساب » يظهر مرتين ، سنعمل منه إجراءاً . المطلوب رؤية كل أرقام الحسابات المتتالية حتى العثور على واحد ينتهي بصفر ؛ إن هذا هو إذن عملية تكرارية من النوع توقُّف مع تضمين والذي يُترجم بـ :

```
repeat lire compte, position, autorisation
until le dernier chiffre du compte est un zéro
```

```
repeat إقرأ حساب ، وضْع ، إذْن
until آخر رقم من الحساب هو صفرا .
```

سيتم كذلك وَصْف الفعل « عالج حساباً » بواسطة إجراءاً وذلك بهدف تأمين قراءة جيّدة للبرنامج ، أخيراً يجب الإشارة إلى الكشوفات أي الحسابات التي يكون الإذْنُ فيها (≥ 0) أقل من الرصيد المدين (< 0) :

```
if position < 0 then
   if abs (position) > autorisation then signaler
```

(position : وَضْع ؛ autorisation : إذْن ؛ signaler . أشِرْ إلى)

```
program decouvert(input,output);
const dernier=0;
var compte:integer;        { رقم الحساب }
    position,autorisation:real;

  procedure lire;
  { ابحث عن الحساب التالي المطلوب فحصه }
  begin
    repeat read(compte,position,autorisation)
    until(compte mod 10)=0
  end ;

  procedure traiter;
  { أشر إلى الحساب المكشوف }
  begin
    if position<0.0 then
      if abs(position)>autorisation then
        writeln('compte',compte:7,'  decouvert=',
                -position-autorisation:13:2)
  end;

begin
  lire;
  while compte<>dernier do begin
    traiter;
    lire
  end
end.
```

(découvert : كَشْف ؛ compte : حساب ؛ traiter ؛ عالج ؛ lire : إقرأ) .

مع المعطيات :

رقم الحساب	وضعه	الإذن
0.00	12.00	1324
100.00	270.50	9710
700.00	− 986.00	0020
100.00	− 200.03	0971
0.00	− 0.27	3640
0.00	0.00	0000

نحصل على : الحساب رقم 20 مكشوف بـ 286.00

الحساب رقم 3640 مكشوف بـ 0.27

2.4.3 ـ موضِعيّ / إجْمالي (Local / global)

لا يمكن إستعمال معرفاً إلاّ

في الفدرة التي تمّ التصريح عنه داخلها : إنه موضعي في هذه الفدرة .

في الفدرات المُتراكِبَة في فدرة التصريح : إنه إجمالي في هذه الفدرات .

هكذا ، مع :

```
procedure P1;
type i ...
var j ...
    procedure P2;
    var k ...
        procedure P3;
        var L ...
        corps de P3
    corps de P2
    procedure P4;
    var m ...
    corps de P4
corps de P1
```

(procédure : إجراء ؛ type : نوع ، var . متغير ؛ corps : جسم) .

- في جسم P1 يمكن إستعمال المتغير j ، الإجراءات P2 وP4 (معرّفين موضعيّين) ، لكن ليس L, P3, K . P1 (إجمالي) يمكن إستعماله .
- في جسم P2 ، يمكن استعمال j, P1 وP2 (معرّفين إجماليين) وكذلك K وP3 (موضعيين) .
- في قسم التصريحات من P2 يمكن إستعمال النوع i (إجمالي) .
- في P3 ، L هو موضعي ؛ K, P2, j, i, P1 و P3 هم إجماليّين .
- في P4 ، m هو موضعي ؛ P4, P2, j, i, P1 هم إجماليين ؛ L (من P3) غير ممكن استعماله .

في الإجراء P4 ، المعرّف l هو بالكامل غير معروف ؛ يمكن إذن التصريح عنه :

```
procedure P4;
var m, L ...
```

فإذن المتغير L من P4 ليس له أية صلة مع المتغير L من P3 .

إذا كان قد تم التصريح عن معرّف في فدرة ، فإنه من الممكن إعادة التصريح عنه في كل فدرة متراكبة في الأولى ؛ ويكون بذلك للمعرّف مدلول آخر :

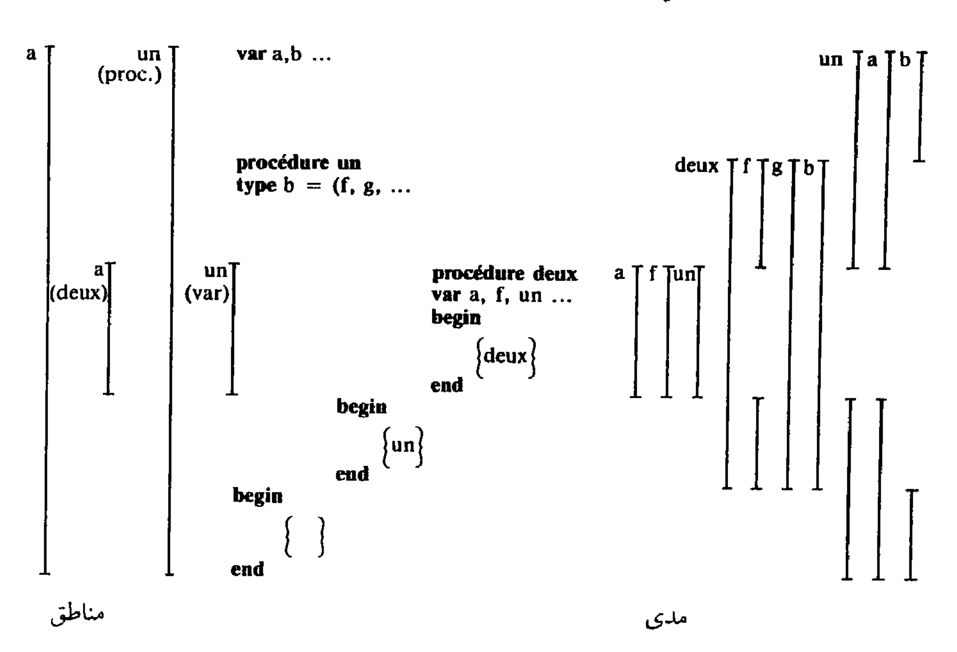
```
...
const a=...
type b=(c,d,e)...
var f,g,h:char...
  procedure P;
  const g=...
  type c=(e,a)...
  var f:char...
```

مثال : في جسم P ، 'f: = 'x لا تؤثر بالمتغير f الإجمالي ؛ بينما 'h: = 'x تؤثر بالمتغير h الإجمالي .

3.4.3 ـ مدى (Portée)

إن مدى تصريح ، أو تعريف هو جزء من البرنامج حيث يكون التصريح صالحاً ، بمعنى آخر القسم الذي يمكن فيه إستعمال المعرِّف مع كل خواصّه المصرّحة . يكون المدى جزء (أو الكل) من منطقته : إنه منطقته مطروح منها مناطق الأسماء المشتركة المحتملة .

- إن منطقة المعرِّف لمتغير ، لنوع ، لثابت ، لإجراء أو لدالة هي فدرة (أقسام التصريحات + قسم العبارات) تصريحه أو تعريفه (تحتوي الفدرة ، الفدرات المتراكبة) .
- لنفترض وجود معرِّف له المنطقة A ؛ إذا كان هناك معرَّف له نفس كتابة الكلمات وعنده المنطقة B الموجودة ضمن A ، فإذن تكون المنطقة B وكل المناطق التي تحويها خارجة عن إطار مدى المعرِّف الأوَّلي :



(حالات خاصة : قائمة الوسائط الصورية ، حقل الفقرة) .

قواعد :

- يجب التصريح أو التعريف (أو التعريف مسبقاً) عن كل معرِّف ؛
- لا يمكن وجود عدة معرِّفين لهم نفس كتابة الكلمات ، مصرَّح عنهم أو معرَّفين في نفس المنطقة ؛

ـ لا يمكن إستعمال المعرِّف إلّا ضمن نطاق مداه ؛

ـ يجب على تصريح أو تعريف المعرِّف أن يسبق كل استعمالاته (ما عدا النوع دليل (pointeur)) (x ، مصرَّحة .... = type x لها كمنطقة : فدرة ، غير أنّ .... = type y : x ; x هي غير سليمة) .

مثال :

```
program portee(output);
var i,j,k,L:integer;
        procedure P1;
        var i,j:integer;
                procedure P2;
                var i,k:integer;
                begin i:=6; k:=6;
                  writeln('P2':5,i:5,j:5,k:5,L:5);
                  i:=2; j:=2; k:=2; L:=2
                end;
        begin i:=5; j:=5;
          writeln('P1':5,i:5,j:5,k:5,L:5);
          P2;
          writeln('P1':5,i:5,j:5,k:5,L:5);
          i:=1; j:=1; k:=1; L:=1
        end;
begin
  i:=0; j:=0; k:=0; L:=0;
  writeln('prog':5,i:5,j:5,k:5,L:5);
  P1;
  writeln('prog':5,i:5,j:5,k:5,L:5);
end.
```

يعطي النتائج :

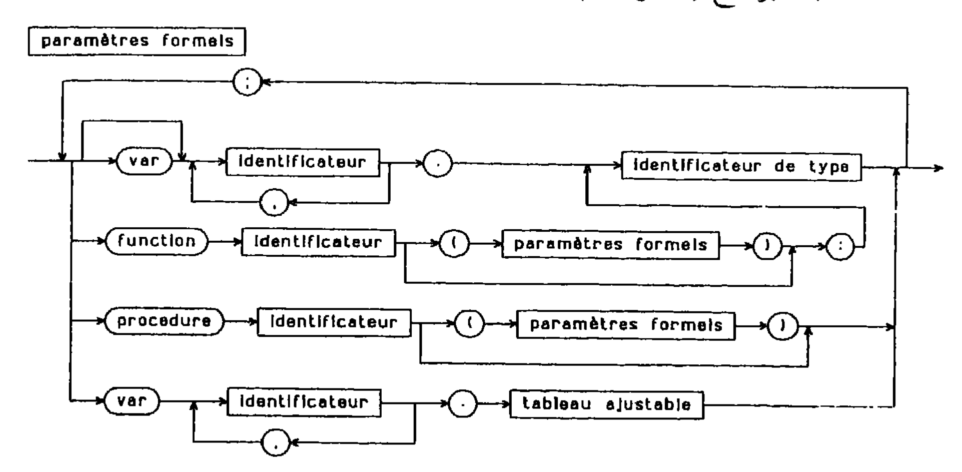
```
0   0   0   0
P1  5   5   0   0
P2  6   5   6   0
P1  5   2   0   2
0   0   1   1
```

4.4.3 ـ وسائط (Paramètres)

يمكن أن يكون للإجراء وسائط بعدد محدَّد معبَّر عنهم في الإجراء بواسطة معرِّف : **الوسيط الصوَري** (paramètre formel) . عند نداء الإجراء ، يجب تحديد **الوسيط الفعلي** ، (paramètre effectif) ، الذي يحلُّ في الإجراء مكان الوسيط الصوَري طيلة فترة تنشيط الإجراء .

إن الطريقة التي بها يَحِلُّ الوسيط الفعلي مكان الوسيط الصوري هي صيغة إنتقاله :

ـ وسيط قيمة : كل إحالة الى الوسيط الصوري هي إحالة الى قيمة الوسيط الفعلي (هذا تعبير) .
ـ وسيط متغيّـر (الكلمة الدليليّـة **Var**) : كل إجالة إلى الوسيط الصوري هي إحالة إلى الوسيط الفعلي ، الذي يجب أن يكون متغيراً .
ـ وسيط إجراء : يكون الوسيط إسماً لإجراء (أنظر 1.4)
ـ وسيط دالّـة : يكون الوسيط إسماً لدالّـة (أنظر 1.4) .
ـ وسيط جدول ضبط : مُنتَقِل بواسطة القيمة أو المتغير ، إنه جدول لا نعرف عدد عناصره لحظة كتابة البرنامج (أنظر 1.4) .



(paramètre formel : وسـيط صُـوَري ؛ identificateur : معـرّف ؛ procédure : إجـراء ؛ tableau ajustable : جدول ضبط) .

وسيط قيمة : يجب أن لا يشتمل نوع الوسيط الصوري على سجل . إن الوسيط الفعلي هو تعبير من نوع متساوق بالنسبـة للتعيين مـع الوسيط الصـوَري . إنه وسيط معطيّـة : ننقل معطيّـة (قيمة تعبير) إلى الإجراء .

وسيط متغيّـر : يكون الوسيط الفعلي متغيراً له نفس نوع الوسيط الصوري . إنه وسيط معطيّـة ونتيجة : إذا كان للمتغير قيمة ، فإن هذه القيمة صالحة لـلاستعمال في الإجراء ؛ إذا عَيَّـن الإجراء قيمة للوسيط ، فإنه يتم نقل هذه القيمة إلى المتغير . يتم تحديد الوسيط قبل تنفيذ الإجراء (النَّيل : دليل ، حقل ، . . . . مُتمِّم) .

مثال :

```
program parametres;
var i,j:char;
  procedure P(k:char;var L:char);
  begin
    writeln ('P',k,L);
    k:=succ(k); L:=succ(L)
  end;
  procedure Q(k:char;var L:char);
  begin
    writeln('Q',k,L);
    P(k,L);
    writeln('Q',k,L)
  end;
begin
  i:='0'; j:='0';
  writeln(i,j);
  Q(i,j);
  writeln(i,j);
  Q(j,i);
  writeln(i,j)
end.
```

يعطي النتائج :

00 ——— **Q (i, j)**
Q00 ——— **P (k, l)**
P00
Q01
01 ——— **Q (j, i)**
Q10 ——— **P (k, l)**
P10
Q11
11

ملاحظة : يمكن أن يكون الوسيط الفعلي المتغير ، سجلاً .

انتبه : يجب أن تتطابق الوسائط الصورية والفعلية من حيث العدد والنوع .

مثال : إجراءاً يحسب مجموع عددين :

```
program P1(input,output);
var a,b,c:real;
  procedure somme(a,b:real; var c:real);   {c=a+b}
  begin c:=a+b end;
begin read(a,b);
  somme(a,b,c); writeln(a,'+',b,'=',c);
  somme(a,1.0,c); writeln(a,'+',1.0,'=',c);
end.
```

(مجموع : somme)

مثال : إجراءاً يحسب مجموع متّجهَيْن من 10 عناصر

```
program P2(input,output);
const n=10;
type indice=1..n;
     vecteur=array [indice] of real;
var a,b,c:vecteur;
    i:indice;
  procedure somme(a,b:vecteur; var c:vecteur);
  var i:indice;
  begin for i:=1 to n do c[i]:=a[i]+b[i] end;
begin
  for i:=1 to n do read(a[i]);
  for i:=1 to n do read(b[i]);
  somme (a,b,c);
  for i:=1 to n do writeln(c[i])
end.
```

(vecteur : متّجِهْ)

مثال : إجراء للفرز

إن الجدول المطلوب فرزه ، معطيّة من الإجراء ، هو كذلك نتيجته : نصرّح عنه كوسيط متغير :

```
type tableau:array [1..n] of ...
   procedure tri(var T:tableau) ...
```

5.3 ـ دَوَالّ (Fonctions)

كالإجراء فإن للدالّة عنوان ، فدرة ، جسم ، وسائط ، لكن

ـ تَنقُلُ الدالة نتيجة ، من نوع بسيط ،

(مثال function f (a, b: integer) : integer تنقل نتيجة صحيحة) .

ـ يتمّ النداء داخل تعبير (أنظر 1.4.2)

(مثال : ... x : = ... * f (12, − 13))

يجب أن تحتوي فدرة الدالّة على عبارة تعيين واحد على الأقل متعلّق بالنتيجة :

énoncé d'affectation

variable

identificateur de fonction

:=

expression

(مثال : $f := (a - b) \star (a + b)$)

(énoncé d'affectation : عبارة تعيين ؛ expression : تعبير ؛ identificateur : معرّف ؛ variable : متغير)

مثال : دالّـة تحسب مجموع عددين :

```
program F1(input,output);
var a,b:real;
  function somme(x,y:real):real;
  begin somme:=x+y end;
begin read(a,b);
  writeln(a,'+',b,'=',somme(a,b));
  writeln(a,'+',1.0,'=',somme(a,1.0));
end.
```

1.5.3 ـ مدى (portée)

داخل عنوان إجـراء أو دالة ، تكون منطقة معرِّفي الوسائط هي فدرة الإجراء أو الدالّـة ؛ يكون إسم الإجراء أو الدالّـة كـما معرّفي النـوع ، داخل الفـدرة الشاملة . تسمّـى هذه القواعد قواعد المدى المنظِّـفة » .

مثال :

bl فدرة شاملة

function *F (* [*A*] *: T1;* **var** [*B*] *: T2) : T3;*

فدرة الدالة

هكذا يمكن إعادة التصريح عن T1 في الدالة : كما إستطاعت قبلاً الفدرة الشاملة استعمال A .

2.5.3 ـ مَفْعُولُ الحافّـة (Effets de bord)

نتكلم عن مفعول الحافة عندما تُغيِّـر دالة أو إجراء ، متغيراً إجمالياً ونعتبر بشكل عام بأن ذلك أضْعَفَ من حسْنِ قراءة ودقة البرامج ؛ لكي نخفِّف من هذه العواقب ، نكتب في ملاحظات كل إجراء ، المتغيرات الإجمالية القابلة للتغيير ، ونمنع كل مفعول حـافة عن الدالّـة .

أ ـ مثال « جيّـد » :

```
var compte:integer;
...
procedure P;  { يؤثر بالمتغير الإجمالي compte ( حساب ) }
begin compte:=compte+1; ... end;
```

(يصلحُ المتغير الإجمالي لعدِّ تنشيطات الإجراء P) .

ب ـ مثال « سيء »

```
var i,j,k:integer;  a:array [0..99] of integer;
...
function f(a,b:integer):integer;
begin f:=(a+b) mod 100; i:=i+1 end;
...
```

(يصلح «i» لعدّ تنشيطات الدالّـة) .

تؤدي العبارة a [f (i, j)] : = f (i, k) + i إلى نتيجة غير محددة : لا نعرف إذا كان يتم تقييم القسم الأيسر من عملية تعيين قبل القسم الأيمن ، أو العكس (يختلف ذلك من حاسوب إلى آخر وليس بالضرورة محدّداً : لا شيء يمنع بأن يكون هذا هو محض صدفة في كل تعيين) .

إذا $i = 2$ ، $j = 7$ و $k = 11$

ـ في الحالة الأولى ، نحصل على :

إلى اليسار : $f(i, j) = 9$ و $i = 3$

إلى اليمين : $f(i, k) = 14$ و $i = 4$

$f(i, k) + i = 18$

يعني $a[9] := 18$

ـ في الحالة الثانية ، نحصل على :

إلى اليمين $f(i, k) = 13$ و $i = 3$

$f(i, k) + i = 16$

إلى اليسار $f(i, j) = 10$ و $i = 4$

يعني $a[10] := 16$

(يمكن كذلك الحصول على

$f(i, k) = 13$ و $i = 3$

$f(i, j) = 10$ و $i = 4$

$f(i, k) + i = 17$

يعني $a[10] = 17$)

6.3 ـ تمارين

1 ـ لنفترض معطي تاريخاً ، على شكل ثلاث قيم صحيحة M, J وA . جِدْ تاريخ اليوم التالي (إنتبه للسنوات الكبيسة) .

2 ـ في نص مؤلف من أحرف كبيرة ، تباعدات ، نهايات أسطر وسمات أخرى ، نودّ عدّ

كل زوج من الأحرف ؛ سيتم تجاهل كل السمات التي ليست هي أحرف كبيرة .

3 ـ في فحص يشتمل على P إمتحانات ، N مرشح حصلوا على علامات على 20 . لكل إمتحان j يتطابق مُعامِل Cj ؛ لا يُقبل المرشح إلّا إذا حصل على معدّل أقله 12 ؛ إذا حصل على معدل بين 10 و12 فسيخضع لإمتحان شفهي للإستلحاق .

إطبع في كل مرة قائمة أرقام المرشحين المقبولين ، ثم قائمة المرشحين المقبولين لإجراء امتحان شفهي ، ثم قائمة المرشحين المرفوضين ، وذلك بالترتيب التنازلي للمعدّلات .

4 ـ مخطّط دَرَجي (histogramme) : وزّع إلى 20 فترة من نفس الحجم ، نتائج عدة قياسات (اعداد حقيقية) ؛ عَدَد القياسات ليس معروفاً مسبقاً .

5 ـ لنفترض معرفة الإحداثيات (x, y, z) لعشرة نقاط في الفضاء (أوقليدي $\mathbb{R}^3$) . جِدْ النقطتين الأقرب .

6 ـ حوّل عدد روماني (c ≥) إلى قيمته العشرية .

7 ـ إنطلاقاً من عدد ليترات الوقود الموضوعة في سيارة بهدف إملاء خزانها ، وترقيم الكيلومترات المبيّن على العدّاد عند ملء خزان السيارة ، إحسب إستهلاك السيارة بين ملئين لخزان السيارة ومعدّل الإستهلاك (بالليترات لكل 100 كيلومتر) .

8 ـ جِدْ كل الأعداد الأوليّة الأصغر من 100 .

9 ـ جزِّء عدد صحيح موجب إلى عوامل أوّلية .

10 ـ جِدْ عدداً صحيحاً ، أصغر من 100 ، والذي يساوي مجموع أرقام مُكَعَّبِه .

11 ـ إحسب القيمة e ، قاعدة اللوغاريتمات ($\ln(e) = 1$) ، عن طريق جمع المتسلسلة $\sum_{n=0}^{\infty} \frac{1}{n!}$ (نوقف عملية الجمع فور الحصول على: دقة محسوبة $< \frac{1}{n \star n!}$)

ملاحظة : n! « تعني » عَامليُّ العدد n ، $n! = \prod_{1}^{n} i.$

12 ـ تم وضع المبلغ x في البنك للحصول على معدّل فائدة مركبة t . ماذا يصبح هذا المبلغ بعد مضي سنة ، سنتين ، . . 20 سنة .

13 ـ يبيّن الجدول التالي العلاقة الرسمية (الضرائبية) بين الفرنك الثابت والفرنك المتداول :

سنة الإكتساب (التملك) أو الإستهلاك (المصاريف)	مُعامل لتطبيقه على سعر الإكتساب أو المصاريف	سنة الإكتساب أو الإستهلاك	معامل لتطبيقه على سعر الإكتساب أو المصاريف
1950	8,18	1967	3,74
1951	6,99	1968	3,58
1952	6,26	1969	3,36
1953	6,33	1970	3,14
1954	6,36	1971	3,03
1955	6,28	1972	2,85
1956	6,16	1973	2,66
1957	6,00	1974	2,34
1958	5,22	1975	2,09
1959	4,92	1976	1,91
1960	4,74	1977	1,75
1961	4,59	1978	1,60
1962	4,38	1979	1,45
1963	4,18	1980	1,27
1964	4,04	1981	1,12
1965	3,94	1982	1,00
1966	3,84		

اكتب الإجراءات التي تردّ على الأسئلة :

- بأي سعرٍ يجب أن نبيع في الـ 1982 مواداً مشتراة بـ x فرنك (في الماضي) في السنة A ، كي « لا نخسر أموالنا » ؟
- كم كان يكلّف في السنة B ، مواداً كلّفت y فرنك في العام 1982 .
- ما هو معدّل التآكل النقدي بين السنتين c وD ‏(D > C) ؟
- في أية سنة كان معدّل التآكل الأكبر ؟

14 ـ إختزال كولاتز (collatz) : لنأخذ عدداً صحيحاً ، لنقسمه بـ 2 إذا كان عدداً زوجياً ؛ لنضربه بـ 3 ونزيد عليه 1 إذا كان عدداً مفرداً ؛ إذا أعدنا الكرة مرات عديدة فإننا سننتهي دائماً بالحصول على 1 . ما هو عدد الإختزالات المطلوبة لكل من الأعداد من 2 إلى 100 ؟ (مثال : 5 : تتطلّب 5 إختزالات : 1, 2, 4, 8, 16, 5) .

15 ـ لعبة الحياة : في الإطار N × N المحدَّد لـ N^2 خليّة، نضع في البداية بعض الشاغلين .
كل خلية (i, j) تجاور 8 أخرى ووحدها $occ_{i,j}$ هي مشغولة .
ننتقل من جيل إلى آخر عن طريق تطبيق القواعد :
1 ـ يعيش شاغلاً لـ (i, j) إذا $occ_{i,j}$ تساوي 2 أو 3 ،

2 ـ يموت شاغل (i, j) إذا كانت $occ_{i,j}$ أقل من 2 (عدد قليل من الناس) أو أكبر من 3 (عدد كبير من الناس) ؛

3 ـ تستقبل الخلية الفارغة (i, j) شاغلاً في حال $occ_{i,j} = 3$

إنطلاقاً من مجموعة سكانية أوليّـة ، قُم بمظاهرة 12 جيلاً متتالياً .

الفصل الرابع

مفاهيم أكثر تقدماً

سيتم في هذا الفصل توضيح وإتمام المفاهيم الأساسية ، للوصول إلى وصف كامل للغة الباسكال . لكي يتم فهم هذا الفصل بشكل أفضل ، فإن معرفة جيـدة للتقنيات الأساسية للباسكال تبدو ضرورية .

لقد تمَّ تجميع مختلف النقاط المعالَجَة تبعاً لمواضيعها : إن قـراءة هذا القسم من الكتاب ليست بالضرورة متتالية .

سنحاول ، من خلال الأمثلة المقدَّمة عامة على برامـج كاملة ، تبيـين كيف يتم تكوين برنامج مركّـب (مهيكل) ، وكيف أن التفكير بالمعطيات والخوارزمات على مستوى مجرّد كفايةً ، يقود عمليات البحث المتتابعة .

1.4 ـ وسائط (paramètres)

يمكن أن تكون الوسائط المنقولة إلى إجراء أو دالة ، من إحدى هذه الصيغ :

ـ وسيط قيمة : الوسيط الفعلي هو قيمة لتعبير (أنظر 4.4.3) ؛

ـ وسيط متغير **(Var)** : الوسيط الفعلي هو متغير ، الذي يُمكِن للإجراء تعديل قيمته (أنظر 4.4.3) .

ـ وسيط إجراء **(Procedure)** : الوسيط الفعلي هو إسم إجراء ؛ ننقل بذلك فعلاً وليس معطيّـة .

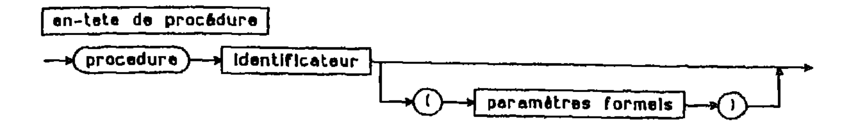
ـ وسيط دالّـة **(function)** : الـوسيط الفعلي هـو إسم دالة ؛ ننقـل بذلـك مؤثراً وليس معطيّـة .

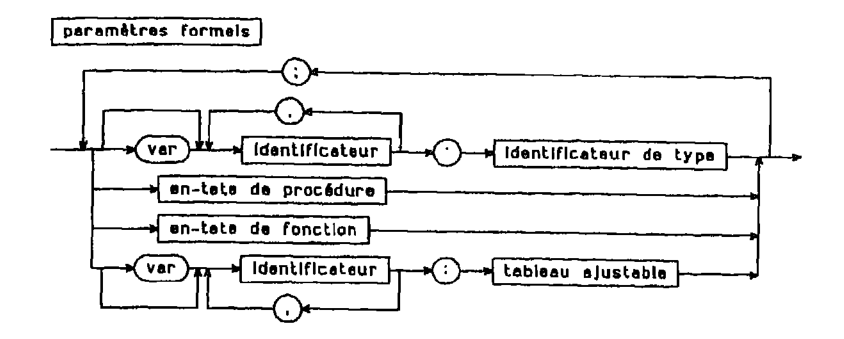
ـ جدول ضبيط(1) ، بالقيمة أو بالمتغير : يعرِض إذ ذاك الوسيط الصوري نموذجاً لجدول

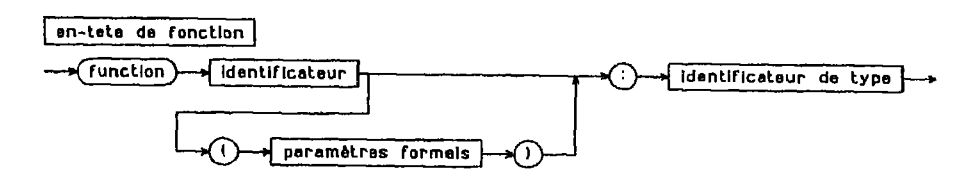
---

(1) تُعرِّف النظم AFNOR مستويين من اللغات ، 0 و1 ؛ يحتوي المستوى 1 إضافة الى المستوى 0 الوسائط جدول ضبيط . لكن كثير من قواعد الباسكال تتماشى مع المستوى 0 ، ولا تعالج إذن الجداول الضبيطة .

يُضبط على الجدول المعطي كوسيط فعلي ؛ يمكن بذلك كتابة إجراء أو دالة يعمل على جدول دون المعرفة المسبّقة لحدود دلائله .







(en-tête : عنــوان ؛ procédure : إجراء ؛ paramètres formels : وسـائط صـورية ؛ identificateur : معرّف ؛ tableau ajustable : جدول ضبط ؛ fonction : دالّة ؛ type : نوع)

1.1.4 ـ وسيط إجراء وسيط دالّة (Paramètre Procédure, Paramètre fonction)

يجب أن يكون الوسيط الفعلي معرِّفاً لإجراء ، أو لدالة ، جرى تعريفه في البرنامج . بالطبع يجب أن تتطابق وسائطه ، إذا وُجِدت ، مع الوسائط المصرَّحة في قائمة الوسائط الصورية : مع

Procedure P (**Procedure** a (i : real; **Var** c: char));

يمكن تسمية P بـ P(B)

إذا كان للإجراء B العنوان :

Procedure B (x : real ; **Var** y : char)

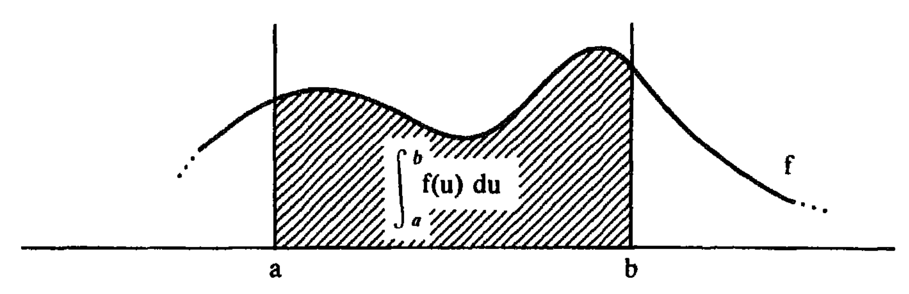
فإذن يمكن في P إستعمال B منقول كوسيط :

a (5.2, d) مثلاً .

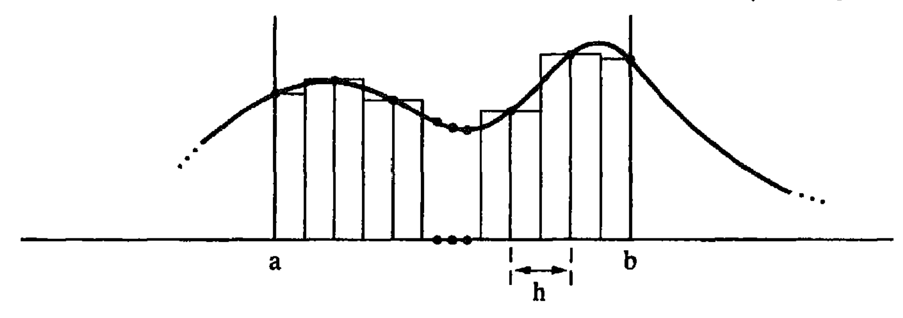
إن الأسماء (c و i) المعطيّة للوسائط الصورية الخاصة بالإجراء المصرَّح a ، ليس لها أي مدلول خاص في باقي البرنامج : إنها لا تصلح إلاّ للتدليل على وجود الوسائط .

مثال : مُكامَلة (intégration)

إن التكامل $\int_a^b f(u)\,du$ لدالة f هو المساحة المحدّدة بمحور الإحداثيات الأول ، المنحنى محدّد بـ a وb :



يمكن إجراء الحساب بطريقة تقريبية وذلك بتقطيع المساحة إلى مستطيلات يمكن بسهولة حساب مساحاتها :



إن مساحة مستطيل مركز على محور الإحداثيات الأول x هو إذن $h * f(x)$

إذا كان يوجد $n - 1$ مستطيل كامل ، ونِصفيْ مستطيل (في a وb) ، فإن القيمة التقريبية للتكامل i هي :

$$\int_a^b f(u)\,du \simeq \left(\sum_{i=1}^{n-1} h \star f(a + i \star h)\right) + h/2 \star f(a) + h/2 \star f(b)$$

حيث $h = (b - a) / n$

$$\int_a^b f(u)\,du \simeq h \star \left((f(a) + f(b))/2 + \sum_{i=1}^{n-1} f(a + i \star h)\right)$$

إذا أخذنا n كبيرة جداً ، فإذن h صغيرة ، نحصل بذلك على قيمة تقريبية للتكامل .

تعطي عملية المكاملة (لدالة حقيقية « جيدة ») نتيجة من نوع بسيط ، حقيقي ؛ يمكن إذن كتابتها على شكل دالة لها كوسائط عددين حقيقيين (الحدود a وb) ، الدالة المطلوب حساب تكاملها (f) ، وعدد الفترات n .

```
function integrale(a,b:real;
                   function f(x:real):real;
                   n:integer)
                             :real;
var V:real;
    h:real;        { (b - a) / n : عرض الفترة }
    i:integer;
begin
  V:=0; h:=(b-a)/n;
  for i:=1 to n-1 do V:=V+f(a+i*h);
  integrale:=h*((f(a)+f(b))/2+V)
end;
```

لنفرض أننا نريد حساب تكامل الدالة e^{-u^2} على الفترة [1- ، 1] مع 1000 خطوة للمُكامَلة :

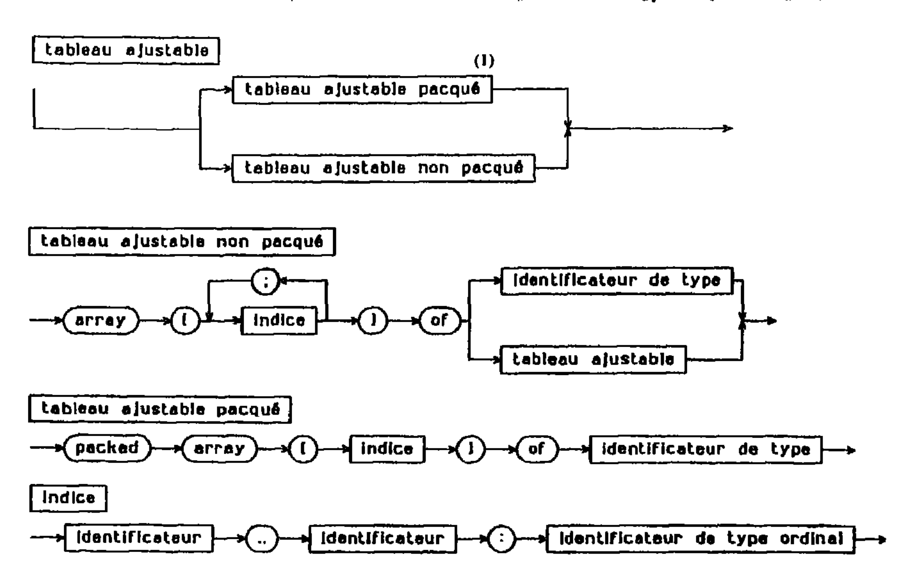
```
program integration(output);
{calcul de "somme de -1 à 1 de e puissance -u2"
 par la méthode des rectangles}

   function integrale(
      a,b:real;                    { حدود المكاملة }
      function f(x:real):real;     { الدالة المطلوب حساب تكاملها }
      n:integer)                   { عدد الخطوات }
                :real;
   var V:real;
       h:real;        { a) (b - / n : عرض الفترة }
       i:integer;
   begin
     V:=0; h:=(b-a)/n;
     for i:=1 to n-1 do V:=V+f(a+i*h);
     integrale:=h*((f(a)+f(b))/2+V)
   end;

   function cloche (t:real):real;
   begin cloche:=exp(-sqr(t)) end;

begin
  writeln('l''integrale de -1 a 1 de e(-t2) vaut',
          integrale(-1,1,cloche,1000):6:3)
end.
```

2.1.4 ـ وسيط جدول ضبيط (Paramètre tableau ajustable)



(tableau ajustable : جـدول ضبيط ؛ Pacqué : معلّـب ؛ indice : دليـل ؛ identificateur : معـرّف ؛ Type ordinal : نوع ترتيبي) .

إنتبه : تنتمي الوسائط جداول ضبيطة إلى المستوى 1 من لغة الباسكال ؛ إنه عنصر اللغة الوحيد في هذا المستوى ، بينما تنتمي العناصر الأخرى إلى المستـوى 0 . لا يمكن إستعمال الوسيط جدول ضبيط إلا إذا كانت نسخة الباسكال المستعملة من المستوى 1 . إن نسخة متطابقة مع المستـوى 0 من النظم (norme) لا يمكن بـأي حال من الأحـوال أن تتضمّـن إمكانية للجداول الضبيطة .

يمثّـل معرِّفي حدود الدليل ، حدود فترة الدليل المطابقة للوسيط الفعـلي ؛ يعطي الوسيط الفعلي حدود دلائله إلى الوسيط الصوَري . يُسْتَعمل معرّفي حدود الدليل في فدرة الإجـراء أو الدالـة . إن الأدوات المُمَثّـلة بـواسـطة هؤلاء المعـرّفـين ليست ثـوابت ولا متغيّـرات . يجب أن يتوافق نوع الوسيط الفعلي مع نموذج الجدول الضبيط .

جدول ضبيط قيمة : تكون قيمة الوسيط الفعلي (تعبـير) منقولـة ؛ لا يمكن أن يحتوي الوسيط الفعلي قبلاً على الوسيط جدول ضبيط (إلاّ في حالتين : في وسيط دالّـة ، في بعض المتغيّـرات الدليليّـة) .

جدول ضبيط متغيّـر : يمثل الوسيط الصوَري الـوسيط الفعلي ، الـذي يجب أن

(1) سيتم توسيع الشرح عن الرص (Packed) في 1.3.4

يكون متغيراً . لا يمكن أن يكون الوسيط الفعلي مركّباً لمتغيّر معلّب (Pacquée) .
اختصار : « ; » يمكن أن تحلّ مكان «] of array [» .

ملاحظة : سيتم بحث الأدوات المعلّبة في 1.3.4 ، يمكن أن يكون الوسيط الفعلي من النوع سلسال (انظر المثال في 2.3.4) ، لكن الوسيط الصوري لا يمكن أن يكون كذلك (ليس معبّر عنه بواسطة النوع جدول) .

مثال جداء حسابي (Produit scalaire)

```
procedure ProduitSalaire (
         var A,B:array[inf..sup] of real; var C:real);
var i:integer;
begin
  c:=0.0;
  for i:=inf to sup do C:=C+A[i]*B[i]
end;
```

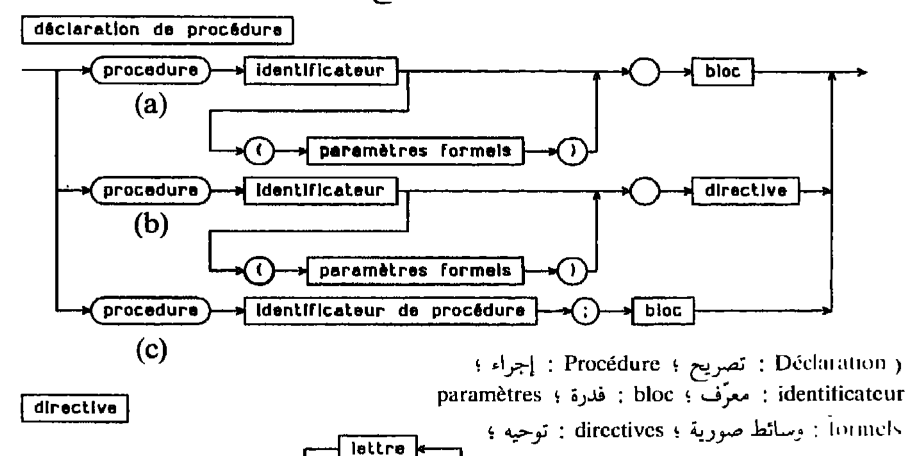
الإستعمال : في برنامجٍ حيث جرى التصريح :

```
type indice = 0..19;
var X,Y:array[indice] of real;
    Z:real;
```

يمكن كتابة : Produit scalaire (x, y, z)

3.1.4 ـ توجيهات (Directives)

يمكن أن يظهر التوجيه إلّا مشتركاً مع عنوانٍ لإجراء أو دالّة ؛ إنه يسمح بفصل العنوان عن الفدرة المقابلة : بدون توجيهات ، تتبع الفدرة العنوان :



(Déclaration : تصريح ؛ Procédure : إجراء ؛ identificateur : معرّف ؛ bloc : فدرة ؛ paramètres formels : وسائط صورية ؛ directives : توجيه ؛ lettre : حرف ؛ chiffre : رقم) .

في الحالة (a) يكون العنوان والفدرة متتاليين .

تتوافق الحالات (b) و(c) مع فصْلٍ للعنوان والفدرة ؛ هكذا مع التوجيه forward (أنظر 2.4) ، الذي هو التوجيه الوحيد المطلوب من قبل النظم ، إلى تصريح العنوان يجب أن يتماشى بعدئذٍ تصريح الفدرة وذلك في نفس قسم التصريحات :

```
(b)     procedure P (i : integer); forward;
        ...
      { procedure P;
      { var ...
(c)   { begin ...
      { ...
      { end;
```

(لا نكرر الوسائط ، المعروفة سابقاً) .

هـذا التنويط يسمـح **بالتكـرارية المتقـاطعة** (récursivité croisée) المشـروحة في 2.4) .

يوجه توجيه آخر في عدد من الحاسوبات : external . إنه يحدد بأن فدرة الإجراء أو الدالة هي خـارج فدرة البـرنـامـج ، أي أن البرنـامج الحـاوي على التصـريح (a) مـع external ، لا يحتوي على التصريح (b) ؛ هذا التصريح (b) قد تم تصريفه على حدة .

بما أنه يجب التصريح عن كل معرِّف قبل إستعماله ، فإن استعمال التوجيه يسمح بتحديد إسم ونوع وسائط إجراء أو دالة ، دون الحاجة الى التحديد المباشر للأفعال .

2.4 ـ التكرارية (Récursivité)

إن تنشيط إجراء أو دالة يحيي المتغيرات المصرَّحة في الفدرة المرافقة ؛ قبل التنشيط ، أو بعد « العودة إلى المنادَى » ، لا وجود لهذه المتغيرات ، أي لا يمكن بلوغهن .

إذا تمّ من جديد تنشيط فدرة نَشِطَة سـابقاً ، فـإن المتغيرات المصـرَّحة تُحيَى من جديد ، وتصبح قِيَم المتغيرات المرافقة للتنشيط القديم غير قابلة للبلوغ (بالرغم من أن هذه المتغيرات الجديدة والقديمة مرتبطة بنفس المعرِّف) ؛ نجد ثانيةً هذه القيم القديمة عند إزالة تنشيط الفدرة الجديدة .

إن التكرارية هي فِعْلُ تنشيط فدرة نشِطة سابقاً ؛ إنه مفهوم قريب من التثنية إلى الوراء (récurrence) . مثلاً يُعرَّف القاسم الأكبر المشترك PGCD لعددين a وb بالتثنية إلى الوراء كما يلي :

حيث n هو الدليل الأول بشكل أنّ $v_n = 0$

$pgcd(a, b) = u_n,$

$\begin{cases} u_i = v_{i-1} \\ v_i = u_{i-1} \textbf{ mod } v_{i-1} \end{cases} \quad \begin{cases} u_0 = a \\ v_0 = b \end{cases}$

مثال :
$u_0 = 25 \quad v_0 = 10$
$u_1 = 10 \quad v_1 = 5$
$u_2 = 5 \quad v_2 = 0 \quad u_n = 5)$

يُكتبُ تعريف عملية الـ pgcd إذن :

$pgcd\ (u, v) = \bar{u}$ **si** $\bar{v} = 0$, $pgcd\ (\bar{v}, \bar{u}$ **mod** $\bar{v})$ **sinon**

(si : إذا ، sinon : وإلاّ)

وفي لغة الباسكال على شكل دالّـة :

```
program PlusGrandCommunDenominateur(input,output);
var a,b:integer;
     function pgcd(u,v:integer):integer;
     begin
       if v=0 then pgcd:=u
       else pgcd:=pgcd(v,u mod v)
     end;
begin
  read(a,b);
  writeln('pgcd(',a:1,',',b:1,')=',pgcd(a,b):1)
end.
```

مع المعطيات a = 25 وb = 10 ، يكون تتالي الحسابات :

التنشيط الأول :
pgcd (25, 10)
u = 25
v = 10

التنشيط الثاني :
pgcd (10, 25 mod 10)
u = 10 25
v = 5 10

التنشيط الثالث :
pgcd (10, 10 mod 5)
u = 5 10 25
v = 0 5 10

تسدى إذن نتيجة التنشيط الثالث 5 (pgcd : = u)

من جديد يأخذ التنشيط الثاني القيمة
u = 10 25,
v = 5 10

(pgcd : = pgcd (v, u mod v)) يساوي 5

التنشيط الثالث

من جديد يأخذ التنشيط الأول u = 25 ، النتيجة تساوي 5 .

v = 10

النتيجة : pgcd (25, 10) = 5

إن الكتابة (...) pgcd = : في دالة من نفس الإسم هي نداء تكراري . لكن يمكن لحساب الـ pgcd أن يُكتبُ أيضاً على شكل تكرارية بسيطة[1] ؛ في المقابل ، فإن حساب دالّـة « أكرمان » (Ackermann) لا يمكن أن يُردَّ بشكل منسّـق إلى تكرارية :

$$\begin{cases} ack\ (0, j) = j + 1 \\ ack\ (i, 0) = ack\ (i - 1, 1) \\ ack\ (i, j) = ack\ (i - 1, ack\ (i, j - 1)) \end{cases}$$

```
program Ackermann(input,output);
var x,y:integer;  compte:integer;

      function ack(i,j:integer):integer;
      begin
        compte:=compte+1;
        if i=0 then
          ack:=j+1
        else
          if j=0 then
            ack:=ack(i-1,1)
          else
            ack:=ack(i-1,ack(i,j-1))
      end;
begin
  read(x,y);
  write('ack(',x:1,',',y:1,')=');
  compte:=0;
  writeln(ack(x,y):1,'   en ',compte,' appels')
end.
```

(إن عبر المتغيّر compte هو مفعول حافة للدالة ack) .

إن التعيين : $ack := ack\ (i - 1, ack\ (i, j - 1))$

يمكن أن يكتب كذلك : $ack := ack\ (i - 1, f\ (i, j))$

حيث $f\ (i, j) = ack\ (i, j - 1)$

[1]
```
while v<>0 do begin r:=u mod v;  u:=v;  v:=r  end;
pgcd:=u;
```

إذا أردنا التصريح عن هاتين الدالتين في نفس المستوى دون أن نُراكِبهم ، فإنه لا يمكننا تصريح f ومن ثم ack ، ذلك لأن f تُنادي ack التي لم تصرَّح حتى الآن ؛ يوجد تكرارية متقاطعة ونستعمل التوجيه forward (أنظر 3.1.4) :

```
function ack (i,j:integer):integer; forward;

function f(i,j:integer):integer;
begin
  f:=ack(i,j-1)
end;

function ack;
begin
  compte:=compte+1;
  if i=0 then
    ack:=j+1
  else
    if j=0 then
      ack:=ack(i-1,1)
    else
      ack:=ack(i-1,f(i,j))
end;
```

(هذا ما يسمح باستعمال الدالّـة f في مكان آخر غير ack) .

3.4 ـ الرصّ ، سلاسل السمات (compactage, chaines de caractères)

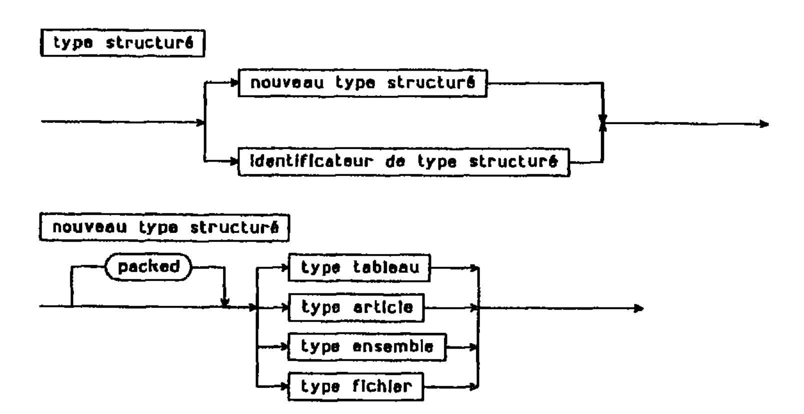
ان الرصّ الذي ينتج عنه تقليل للأحجام في الذاكرة ، هو ضروري في حالتين :

ـ للتقليل من الحجم في الذاكرة لبرنامج يُعالج متغيرات مركّـبة ؛ الأداة 0...] of char [illegible] array تحتلّ 10000 كلمة في كثير من الآلات ؛ عند رصّـها بـ 0...] of char [illegible] **Packed array** فإنها لا تحتل ، تبعاً للآلة ، إلاّ 5000 ، 2500] (IBM 370) أو حتى 1000 (CDC Cyber) ؛

ـ لتسهيل معالجة النصوص ؛ إذا كان قد تم تصريح **array** [1..3] **of char** C يجب كتابة [illegible] C [1] : = 'T' ؛ C [2] : = 'R' ، C [3] : = 'I' أو (C [1] = 'T') **and** ([illegible] = 'R') **and** (C [3] = 'I') بينما إذا تم تصريح **Packed array** [1..3] **of char** ، فان C تعالَج بـ C : = 'TRI' أو C = 'TRI' .

1.3.4 ـ الرصّ (Compactage)

إن ظهور الرمز **Packed** في تعريف من نوع مركّـب يعني أن هذا النوع هو مُعلَّـب ؛ إذن حجم القيَم في الذاكرة خُفـض إلى أقـل قدر ممكن ، في مقـابل خسـارة في فعاليـة العمليات على هذه المتغيرات فيما خص الوقت وكذلك المكان (للبرنامج) .



(type structuré : نـوع مركّـب ؛ nouveau : جـديد ؛ identificateur : معـرّف ؛ tableau : جـدول ؛ article : فقرة ؛ ensemble : مجموعة ؛ fichier : سجل) .

في نوع معلّب ، لا يعلّب مركّب الذي يكون هو نفسه مركّب إلا إذا أشير إليه صراحة معلّب .

تبعات الرصّ :

- إن نوع جدول معلّب ، ذي دليل منطلق من 1 ، ومحتوياً لسمات ، هو نوع سلسال ذو خصائص محددة ؛
- إن وسيطاً فعلياً متغيراً لا يمكن أن يكون مركّباً لمتغير معلّب ؛
- لكن يمكن لوسيط من الإجراء read (إذن readln) أن يكون مركّباً لمتغيّر معلّب ؛ كذلك فإن وسيطاً من الإجراء write (أو writeln) الذي هو تعبير ، يمكن أن يكون مركباً لمتغيّر معلّب .

إن منفذاً إلى مركّب لمتغير معلّب يمكن أن يظهر في كل تعبير ، وفي تعيين ، في القسم الأيسر أو الأيمن . بالمقابل ، فإن نوعاً معلّباً ونوعاً غير معلّب ليسا متساوقان بالنسبة للتعيين (إنهما ليسا من نفس النوع) ؛ يمكن إذن إستعمال إمّا تكراريّة لتعيينهم مركّب بمركّب ، إمّا استعمال الإجراءات المعرّفة سلفاً Pack وunpack .

لنفرض التصريح

var *a*: **array** *[s1]* **of** *T*;
    z : **packed array** *[u...v]* **of** *T*;

- يسمح الإجراء **pack** بالرصّ :

pack (a, i, z) مكافئ لـ :

```
begin k:=i;
      for j:=u to v do begin
           z[j]:=a[k];
           if j<>v then k:=succ(4)
      end
end
```

ـ يسمح الإجراء **unpack** بانفضاض الرصّ :

unpack (z, a, i) مكافىء لِـ

```
begin k:=i;
      for j:=u to v do begin
           a[k]:=z[j];
           if j<>v then k:=succ(k)
     end
end
```

حيث j وk هما متغيران مساعدان .

2.3.4 ـ سلاسل (Chaines)

كل نوع **Packed array** [1..n] of char ، حيث تكون n > 1 ، هو نوع سلسال من n سمة . إن ثابتاً 'xxx' من n سمة هو كذلك من النوع سلسال من n سمة ؛ يصبح إذن التعبير إلى متغير من نفس النوع ممكناً .

الخصائص :

ـ يمكن أن يظهر المتغير أو الثابت ، من النوع سلسال في قائمة وسائط write ؛

ـ نطبق مؤثرات العلاقة .

= <> < <= >= >

على النوع سلسال ، تبعاً للترتيب المعجمي المحدّد بلُعب السمات ؛ يجب أن تكون القيمتين الموصولتين بالعلاقة متساوقتين ، أي لهما نفس عدد السمات . (من المفيد الملاحظة بأن العلاقات لا تطبق على النوع جدول) . تتعلق قيمة تعبير مثل c < 'IL Y A' بلعب السمات المستعمل . .

لا يمكن لمتغير من النوع سلسال أن يكون وسيطاً لـ read ؛ يجب إستعمال عبارة مثل :

```
for i:=1 to N do read(V[i])
```

حيث تمّ التصريح عن V

var *V* : **packed array** *[1..N]* **of** *char*

إن إجراءاً عاماً لقراءة السلاسل يُكتب :

```
procedure lireChaine (
          var X:packed array[un..max:integer] of char);
label 1;
var car:integer;
begin
  for car:=un to max do
    if eof then begin
      writeln('eof lors d''une lecture de chaine')
      goto 1
    end
    else
      read(X[car]);
1:end;
```

إنه من السهل إذن كتابة إجراءات تجري عمليات على سلاسل : تعيين سلاسل (ذات أطوال مختلفة) ، تنضيد (concaténation) ، إستخلاص (extraction) ، . . .

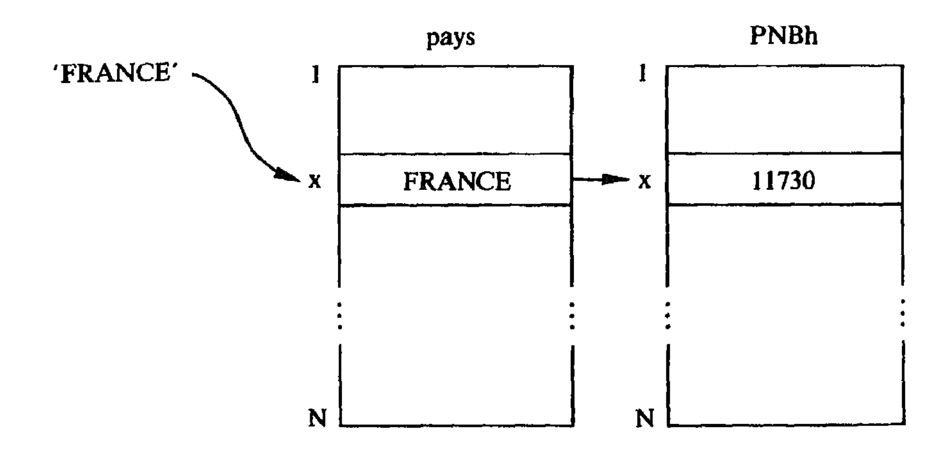
مثال : البحث في جدول من السمات

إكتب برنامجاً يسمح بالبحث عن قيمة الإنتاج القومي الخام (PNB) للفرد (بالدولار) ، لبلد معين .

يجب تهيئة جدول مطابقة « بلد ← PNB » للفرد والذي سيتم وصفه في الباسكال بواسطة المتغيرات :

```
pays: array[1..N] of packed array[1..l] of char;
PNBh: array[1..N] of integer;
```

إن موقع إسم البلد في الجدول الأوّل ، يعطي في الجدول الثاني قيمة الإنتاج القومي الخام للفرد فيه (PNBh) :



فاذن نحصل على الصيغة الأولية :

1 ـ

```
1) program recherche(input,output);
   { البحث عن الـ PNB للفرد لبلد معين }
   const N=?; L=?;
   type chaine=packed array[1..L] of char;
   var pays: array[1..N] of chaine
       PNBh: array[1..N] of integer;
       nom: chaine;
   begin
     initialiser les tableaux pays et PNBh
     repeter
       demander un non ( اطلب إسماً )
       calculer x son indice dans ( احسب دليله في الجدول pays )
       et écrire pNBh[x] PNBh [ x ] واكتب
     en excluant le dernier nom: ´fin´
   end.          ( لا تأخذ بالحسبان الإسم الاخير 'fin' )
```

2 ـ « دمّث الجداول Pays (بلد) وPNBh » . نقرِّر بأنه سيوجد N سطر معطيات ، كل واحد منه مؤلف من عدد صحيح ومن سلسال ؛

```
procedure initialiser;  الإجراء دمّث
var ligne : integer;
begin
  for ligne:=1 to N do
     ( إقرأ PNBh [ سطر ] و
     Pays [ سطر ] وانتقل إلى السطر التالي ])
end;
```

3 ـ « إقرأ PNBh [سطر] وPays [سطر] ، وانتقل إلى السطر التالي » تكتب :

```
read(PNBh[ligne]);
lire pays[ligne] ( إقرأ Pays [ سطر ] )
readln
```

4 ـ إقرأ Pays [سطر] ، يعني قراءة سلسال غير كامل :

```
procedure lireChaine(var c:chaine);
var i,j:integer;
begin
  i:=0;
  while not eoln do begin  i:=i+1; read(c[i])  end;
  for j:=i+1 to L do c[j]:=´ ´;
  readln
end;
```

إذا أهملنا الحالات التي يكون فيها السلسال المطلوب قراءته طويلاً ، أو متوقفاً على نهاية سجل .

5 ـ «répéter .. en excluant» (كرّر . . . لا تأخذ بالحسبان) تُترجم بـواسطة عبارة while مسبوقة بالحصول على أول عنصر مطلوب معالجته :

```
demander un nom   ( اطلب إسما )
while nom<>'fin' do begin
  calculer x son indice dans le tableau pays, et
  écrire PNBh[x]          ( إحسب x دليله في الجدول Pays ) وإكتب PNBh [ x ]
  demander un nom
end;
```

6 ـ « أطلب إسما » هو نداء لإجراء سبق تعريفه :

lire chaine (nom)

7 ـ « أحسب x دليله في الجدول pays ، . . . » يمكن ترجمته بعدة أشكال :

ـ إذا احتوى الجدول على الأسماء بحالة غير مرتّبة ، نجري عملية بحث متتالية :

```
x:=0;
repeat  x:=x+1  until (pays[x]=nom) or (x=N);
if pays[x]<>nom then writeln('pays inconnu: ',nom)
else writeln('PNB/hab(',nom,')=',PNBh[x]);
```

ـ إذا كان الجدول مرتباً ، يمكن تسريع عملية التنقيب (dichotomie فُرقان)

لقد اتّبعنا طريقة التدقيق المتتالي : كل فِعلٍ معقد هو مركب من عدة أفعال التي يمكن كتابتها مباشرة ، إذا كانت بسيطة ، أو وصفها بطريقة عديمة الشكل بواسطة جمل فرنسية لكي يتم تشريحها فيما بعد .

تجمع البرنامج الكامل كل هذه المراحل :

```
program recherche(input,output);
{ البحث عن PNB للفرد في بلد معين }
( الفائمة الأولية للبلدان غير مرتّبة )
const  N=16;          { عدد البلدان }
       L=15;          { الطول الأقصى للإسم }
       fin='fin            ';
type chaine=packed array [1..L] of char;
var pays:array [1..N] of chaine;      { قائمة البلدان }
    PNBh:array [1..N] of integer;
    nom:chaine;
    x:integer;        { دليل في الجداول Pays وPNBh }

  procedure lireChaine (var c:chaine);
  var i,j:integer;
  begin
    i:=0;
    while not eoln do begin  i:=i+1; read (c[i])  end;
    for j:=i+1 to L do c[j]:=' ';
```

```
    readln
  end;

  procedure initialiser;
  var ligne:integer;
  begin
    for ligne:=1 to N do begin
      read(PNBh[ligne]); lireChaine(pays[ligne])
    end
  end;

begin  initialiser;
  lireChaine(nom);
  while nom<>fin do begin
    x:=0;
    repeat  x:=x+1  until (pays[x]=nom) or (x=N);
    if pays[x]<>nom then writeln('pays inconnu: ',nom)
    else writeln('PNB/hab(',nom,')=',PNBh[x]);
    lireChaine (nom)
  end
end.
```

المعطيات الأولية

```
11730france
12180belgique
13590allemagne
7920grande-bretagne
13520suede
11360etats-unis
9890japon
11330islande
4880irlande
16440suisse
1460turquie
26080qatar
270haiti
1930roumanie
6710rda
80bouthan
suisse
france
usa
etats-unis
fin
```

النتائج :

```
PNB/hab(suisse          )=16440
PNB/hab(france          )=11730
pays inconnu: usa
P .B/hab(etats-unis     )=11360
```

عندما يتم فرز جدول أسماء البلدان ، يمكن إجراء التنقيب كما الحال مع القاموس : نجرّب في الوسط ؛ إذا كان الإسم المطلوب قبل ، نعيد إجراء عملية التنقيب في القسم الأول ، وإلاّ في القسم الثاني :

التنقيب في الفترة a ... b =

التنقيب في $\frac{a+b}{2}$ إذا بلد $\left[\frac{a+b}{2}\right]$ < الإسم ، وإلاّ

التنقبب في .. $\frac{a+b}{2}$

« إحسب x دليله . . . » تُكتب إذن :

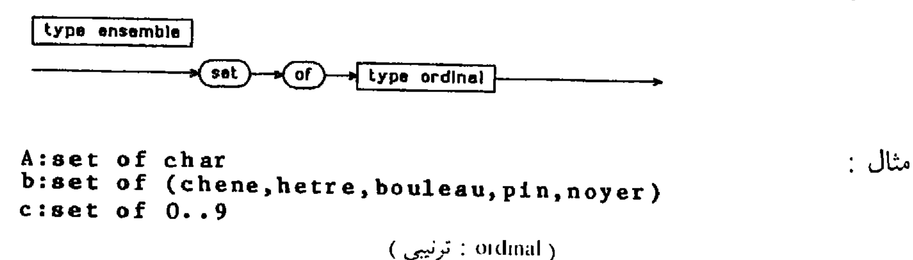
```
u:=1; b:=N+1;
repeat
  c:=(a+b)div2;
  if nom<pays[c] then b:=c else a:=c
until b<=(a+1);
if nom=pays[c] then writeln(PNBh[c])
else writeln('paysinconnu');
```

(بلد مجهول : Pays inconnu)

هذه الطريقة المسماة بالتنقيب الفرقاني تصبح مـلائمة لِـ N = عـدد كبير (عـدد المقارنات متناسب مع $\log_2 N$ ، بينـما بالنسبـة للتنقيب المتتالي يكـون تقريبـاً مساويـاً لِـ N/ 2) .

4.4 ـ مجموعات (ensembles)

لكي نعرف إذا كانت قيمة متغير من النوع سمة هي التكويد لرقم ، يمكن بالطبع كتابة ('9' = < x) and ('0' = < x) ، لكن من الأسهل إستعمال فكرة المجموعة : ['0'... '9'] x in . تُكوَّن المجموعة إنطلاقاً من نوع قاعدي (ترتيبي ، هذا ما يستبعد النـوع حقيقي) :



مثال :

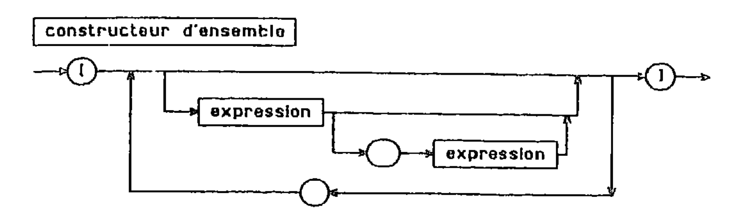
```
A:set of char
b:set of (chene,hetre,bouleau,pin,noyer)
c:set of 0..9
```

(ترتيبي : ordinal)

كل قيمة لـ B مؤلفة من قيم وحيدة من النوع القاعدي :

```
b:=[chene,bouleau,pin]
```

يمكن أن تُكوَّن قيمة من نوع مجموعة بواسطة مُكوِّن مجموعة ، الذي هو عاملٌ لتعبير (أنظر 1.3) :



مثال :

```
if x in ['A','E','I','O','U','0'..'9'] then
  VoyelleOuChiffre
```

(constructeur d'ensemble : مكوّن مجموعة ، expression : تعبير)

بالنسبة لكل الأنواع ـ مجموعات ، تنوّط المجموعة فراغ بـ []

لا يمكن إستعمال القيمة التي يمثلها مكوِّن مجموعة ، في عمليات أو تعيينات إلاّ مع مجموعات من نوع متساوق ، يمكن كتابة ['0', '1', '2'] = : A لكن ليس [0, 1, 2] = : A .

ملاحظة : إن المجموعة المعرَّفة في لغة الباسكال هي من الناحية الرياضيّة مجموعة أجزاء النوع القاعدي . على كثير من الآلات ، تُمثَّل المجموعة بواسطة سلسال من البتة ، هذا ما يفرض قيودا على النوع القاعدي : عدد عناصر محدد ، وقيمة دنيا (صفر) .

إنّ العمليات على المجموعات هي :

+	union إحاد	$A \cup B$	المناثريْن و النسخه هم من نفس النوع
–	différence فرق	$A - B$	
☆	intersection تقاطع	$A \cap B$	
=	égalité تساوٍ	$A = B$	المناثرين هم مجموعات من نفس النوع ، النتيجة هي بولية
< >	inégalité متباين	$A \neq B$	
< =	inclusion (واسع) تضمين	A dans B : A < = B	
> =	inclusion (واسع) تضمين	B dans A : A > = B	
in	appartenance ينتمي	$x \in A$	١ من النوع القاعدي للمجموعة A ، نتيجة بولية

لا تطبّق الإجراءات read وwrite على المجموعات ، لكن من الممكن كتابتها لكل نوع مجموعة من الأعداد الصحيحة أو من السمات :

```
type base= 0..31;
     ens= set of base;

procedure writeSet(S:ens);
var x:base; suivant:boolean;
begin
  write('['); suivant:=false;
  for x:=0 to 31 do
    if x in S then begin
      if suivant then write(',',x:1) else write(x:1);
      suivant:=true
    end;
  write(']')
end;

procedure readSet(var S:ens);
var c:char; x:integer;
begin S:=[];
  read(c);
  if c='[' then begin read (c);
    repeat  {lire un nombre}
      x:=0;
      if c in ['0'..'9'] then begin
        while c in ['0'..'9'] do begin
          x:=x*10+ord(c)-ord('0'); read (c) end;
        if x in [0..31] then S:=S+[x]
        else writeln('erreur: hors intervalle')
      end
```

```
      else
        if c<>']' then
          writeln('erreur: caractere illegal');
      if c=',' then read(c)
      else
        if c<>']' then begin
          writeln ('erreur: ] attendu'); c:=']' end
    until c=']'
  end
  else writeln('erreur: [ attendu')
end;
```

(غلط : erreur ، خارج الفترة : hors intervalle ؛ سمة محظورة : Caractere illégal ؛ متطر : attendu)

5.4 ـ فقرات مع مشتقات ، عبارة مع (Articles avec Variante, Enoncé Avec)

يتم تمثيل أداة ذات عدة مركّبات من أنواع مختلفة بواسطة فقرة . مثلاً يمكن أن تكون الدائرة :

record *centre : xy; rayon : real* **end** (avec **type** *xy* = **record** *x, y : real* **end**)

(مركز : centre ؛ شعاع : rayon)

وقطعة خطٍ مستقيم :

record *origine, extrémité : xy* **end**

(نقطه البدء : origine ؛ طرف : extrémité)

لكن إذا أردنا وصف شكل هندسي ، وَجَبَ تحديد شكل الخط (عادي ، منقوط . .) ولونه (أسود ، غير مرئي . . .) ، وكذلك احداثيات الشكل ؛ إذا كان دائرة فإنّ احداثيات القطعة غير مفيدة :

<table>
<tr><td>خط (trait) :</td><td colspan="2">(عادي ، منقوط)</td></tr>
<tr><td>لون (couleur) :</td><td colspan="2">(أسود ، غير مرئي)</td></tr>
<tr><td>أداة (objet) :</td><td colspan="2">(دائرة ، قطعة ، نقطة)</td></tr>
<tr><td>أداة =
نقطة (point)</td><td>أداة =
قطعة (segment)</td><td>أداة =
دائرة (cercle)</td></tr>
<tr><td>احداثيات : xy</td><td>نقطة البدء
طرف : x, y</td><td>مركز : xy
شعاع : real</td></tr>
</table>

إن الحقول التي يعتمد وجـودها عـلى المبيِّـن (indicateur) أداة (objet) ، سيتم كتابتها في لغة الباسكال في مشتقات فقرة :

```
type xy= record x,y:real end;
     typeObjet= (cercle,segment,point);
     figure=record trait:   (normal,pointille);
                  couleur: (noir,invisible);
                  case objet:typeObjet of
                   cercle:  (centre:xy; rayon:real);
                   segment: (origine,extremite:xy);
                   point:   (coord:xy)
           end;
```

إن الكتابة

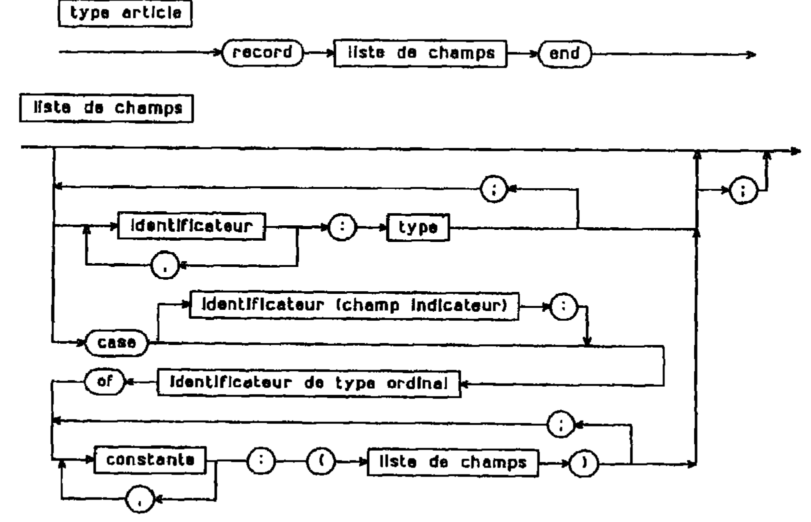
```
case objet:typeObjet of
     cercle:...
```

هي إختصار لـ

```
objet:typeObjet;
case typeObjet of
     cercle:...
```

الحقل objet هو حقل مبيِّـن ، الذي تسمح قيمته بانتقاء لُمشتق . وَجب أن تكون لديه قيمة قبل أن يَبلُغ حقلاً للمشتق .

إذا ساوى الحقل المبيِّـن نقطة (point) ، فوحده الحقل coord (إحداثيات) يمكن بلوغه (مركز ، . . . ، طرف يمكن بلوغهم) ، هذا ما هو مطابقاً لما أردنا وصفه .



(type article : نـوع فقـرة ؛ liste de champ · قـائمـة الحقـول ؛ identificateur : معـرّف ؛ champ : حقل مبيَّـن ؛ indicateur : نوع ترتيبي ؛ type ordinal : ثابت ؛ constante) .

يظهر القسم المنتقى في نهاية وصف الفقرة ، بعد القسم الثابت إذا وُجد . قبل بلوغ حتما لمنتقى يجب إختبار قيمة الحقل المبيَّـن ، يمكن أن تكون عبارة « الحالة » مفيدة :

مع `var F:figure` تمكن كتابة

```
case F.objet of
  cercle: distance:=abs(sqrt(sqr(F.centre.x)+
                    sqr(F.centre.y))-F.rayon);
  segment:  ...
  point: distance:=sqrt(sqr(F.coord.x)+sqr(F.coord.y))
end
```

وكذلك

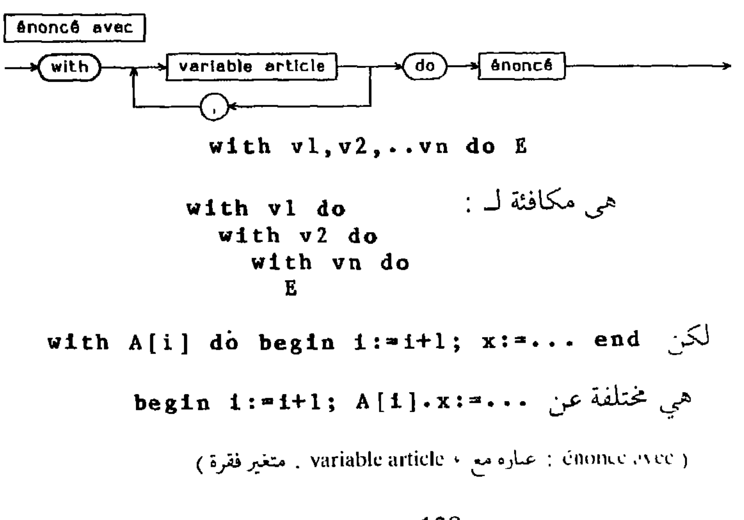
```
F.trait:=normal;    F.couleur:=noir;
F.objet:=cercle;   F.centre.x:=12.7;    F.centre.y:=-0.9;
F.rayon:=1.0;
```

هذا ما يمكن كتابته بطريقة مختصرة بواسطة العبارة مع (Avec) :

```
with F do begin
  trait:=normal; couleur:noir;
  objet:=cercle; rayon:=1.0;
  with centre do begin x=12.7; y:=-0.9 end end
```

(distance : مسافة ، coord . إحداثيات)

إن بلوغ المتغير فقره وِفقا لِـ **with** تتم قبل تنفيذ العبارة وِفقاً لِـ **do** وهذا البلوغ يوجِدُ إسنادا للمتغير طيلة مدة تنفيذ العبارة .




```
with v1,v2,..vn do E
```

هي مكافئة لـ :

```
with v1 do
  with v2 do
    with vn do
      E
```

لكن `with A[i] do begin i:=i+1; x:=... end`

هي مختلفة عن `begin i:=i+1; A[i].x:=...`

(énoncé avec : عبارة مع ، variable article . متغير فقرة)

6.4 ـ أدِلّاء ومتغيرات تحريكية (Pointeurs et Variables dynamiques)

في حالات عديدة ، يمكن زيادة سرعة الحساب ، أو حتى إستبداله عن طريق تركيب ملائم للمعطبات . لنفرض مثلا قائمة رابحي دوري فرنسا للدراجات والأسئلة :

(أ) ـ من ربح في السنة x ؟

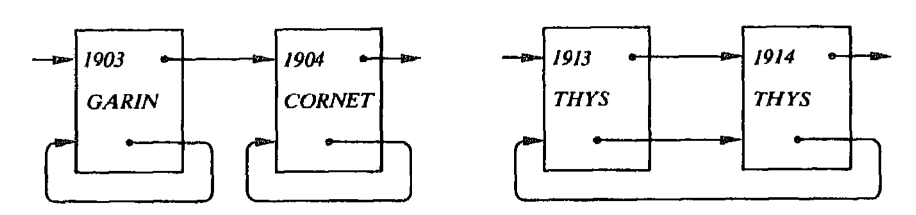
(ب) ـ هل ربح مرات أخرى ؟

(ج) ـ في أية سنة ربح فلان ؟ إلخ . .

إن تمثيلا للمعطيات في جدول

(دوري) tour : array [1903 .. 1990] of packed array [1..n]of char

يسمح بإعطاء جواب سهل على السؤال (أ) (tour [x]) ، شرط إعتماد إصطلاح معيّـن بالنسبة للسنين التي لا رابح فيها، لكنه يُجبر على تصفّـح كل الجدول للإجابة على (ب) و(ج) . إن تمثيلاً مناسباً أكثر للأسئلة يمكن أن يكون على الشكل :



(GARIN, CORNET, THYS هم أسماء الرابحين في السنين المذكورة)

ويمكن وصفه في لغة الباسكال :

```
tour= array [0..87] of record
         annee:1903..1990;
         nom: packed array[1..n] of char;
         autre:-1..87
      end
```

(annee : سنة ؛ nom : إسم ؛ autre : مختلف)

0	1903	GARIN	−1
1	1904	CORNET	-1
2	1913	THYS	3
3	1914	THYS	2

لكن تركيب المعطيات هذا ، لا يسمح بسهولة بزيادة فقرة (1920- THYS) ، ويحجبُ بالأخص الفكرة الأولية .

يمكن وصف العلاقة بين الفقرات بواسطة أَدِلّاء : يسمح الدليل (pointeur) بتسمية أداة . نُفرِّق :

- إسم الدليل ، إنه معرِّف لمتغيّر ؛
- قيمة الدليل : P
- الأداة المدلّل عليها : $P\uparrow$

يكون المتغيّر الدليل مُرتبط بنوع واحد :

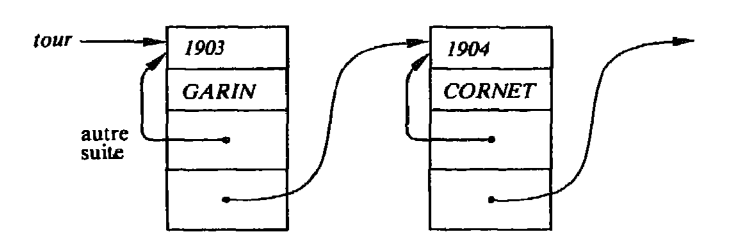
```
type lien=^vainqueur;
     vainqueur=record annee:1903..1990;
                      nom:packed array[1..n] of char;
                      autre : lien
                      suite : lien
                 end;
var tour : lien;
```

(vainqueur : منتصر ؛ lien : رِباط ؛ suite . تابع) .

إنّ Var tour : lien
تتناسب مع التركيب المطلوب . يوجد علاقتين بين الفقرات :

- autre (مختلف) ، يدلّل على إنتصار لنفس المتسابق ،
- unité (وَحْدة) ، تدلّل على السنة التالية .

يدلّل المتغيّر tour (دوري) على رأس قائمة المنتصرين .

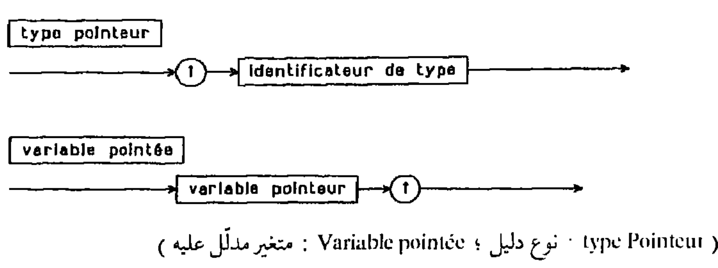


tour	هو من النوع lien
tour↑	هو من النوع Vainqueur
tour↑.année	تساوي 1903
tour↑.autre↑	له قيمة tour ↑
tour↑.suite↑.nom	يساوي « CORNET »

لا يتم التصريح عن الفقرات المدلّل عليها كمتغيرات ، في القسم **Var** ، إنها متغيرات تحريكية ، الذي يجب خلقها صراحة (إجراء new) والذي يمكن إتلافها (إجراء dispose) .

new [p] يخلق منغيراً جديداً من نوع مدلّل عليه بـ p ويعطي للدليل p قيمة تسمح بإسناد المتغير . يمكن إختبار مساواة دليلين مدلّلين على نفس النوع : العمليات = و <> ، إستعمل دليلاً في عملية تعيين (في القسم الأيسر أو الأيمن) وفي تعبير ، وحوّله إلى وسيط . إن القيمة **nil** المُتساوقة مع كل نوع دليل ، تشير بأن دليلاً لا يؤمّن اسناداً لمتغير (إذا p : = **nil** ، يمكن بلوغ قيمة p (لكن ليس ↑ p) ، بينما لا يمكن بلوغ قيمة p إذا كانت غير محددة قبل أية عملية تعيين) .

dispose (p) يُزيل المتغير المدلّل عليه ؛ يعدّ خطأً إذا كـان لـ p القيمة **nil** أو هو غير محدّداً .



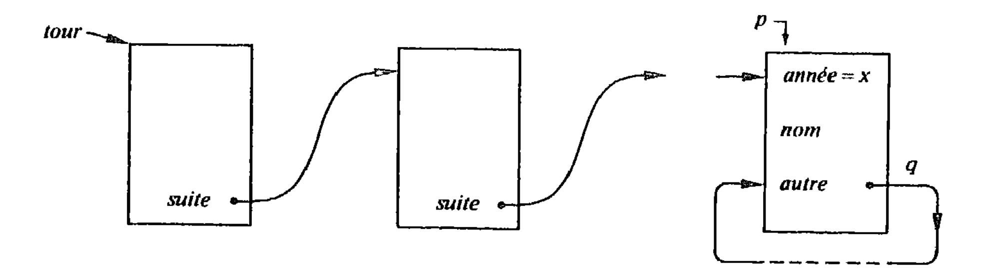
(متغير مدلّل عليه : Variable pointée ؛ نوع دليل : type Pointeur)

يُعرَّف النوع دليل قبل النوع المدلّل عليه (إنه الشواذ الوحيد عن القاعدة « على كل معرِّف أن يصرّح عنه قبل إستعماله ») .

ستتم الإجابة عن السؤال « من ربح في السنة x ، هل ربح مرات أخرى ؟ » بواسطة : (مع **var** p.q: lien ، وعلى أساس أن القائمة مرتّبة بالتنظيم التصاعدي للسنوات) .

```
p:=tour;
while (p^.annee<x) and (p^.suite<>nil) do p:=p^.suite;
if p^.annee=x then begin
  writeln(p^.nom,' a gagne en ',p^.annee);
  q:=p^.autre;
  if q<>p do begin
    write('il a aussi gagne en ');
    while q<>p do begin
      write(q^.annee:5); q:=q^.autre end;
    writeln
  end
end
else writeln ('annee non referencee')
```

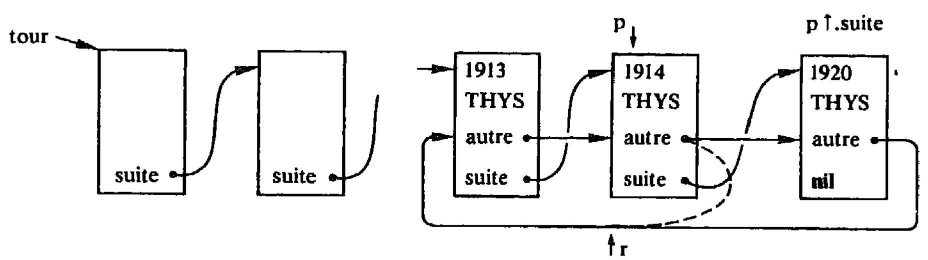
(غير مسندة : non reference ؛ سنة : année ؛ كذلك : aussi ؛ ربح في : a gagné en)



بما أن القائمة تم تشكيلها ، فإن زيادة إسناد جديد لاحق على القائمة (سنة x ، إسم y) سيكون :

(مع var p, q, r : lien) .

```
p:=tour; q:=p; r:=nil;
while q<> nil do begin { ابحث عن p = نهاية القائمة }
  if q^.nom=y then r:=q;
  p:=y;
  q:=q^.suite end;
new(p^.suite);
with p^.suite  do begin  { شكّل قائمة الإنتصارات الأخرى }
  annee:=x;   nom:=y;     suite:=nil;
  if r=nil then
    autre:=p^.suite
  else begin
    autre:=r^.autre; r^.autre:=p^.suite end
end
```



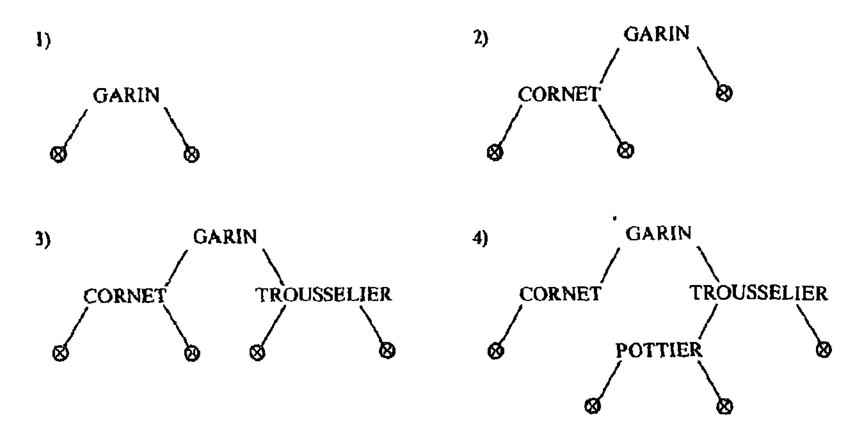
new (p, c1, ... cn) يخلق متغيراً جديداً من النوع فقرة مدلّل عليها ، عن طريق إنتقاء الأقسام المشتقة المعلّبة التي تتطابق مع الثوابت c1 .. cn .

dispose (p, c1... cn) يزيله ؛ يجب أن تكون المشتقات المشار إليها هي نفسها التي كانت عند الخلق .

مثال : فرز ثنائي (tri binaire)

تتلخص الطريقة بالحصول على تكوين شجراني حيث تكون القيمة الموضوعة في عقدة أكبر من كل تلك التي تكون على اليسار وأصغر من كل تلك التي تكون على اليمين ؛

لنفرض أننا نريد فرز 'GARIN' ، 'CORNET' ، 'TROUSSELIER' ، 'POTTIER' ، 'PETIT-BRETON' ، 'FABER' ، إلخ . . .



تتمثَّل العلاقة بين العُقَد بسهولة بواسطة أدِلّاء :

```
type fils=^noeud;
     noeud=record valeur:packed array [1..n] of char;
                  gauche,droite:fils  end;
var racine:fils;
```

إطبع القائمة المفروزة ، يعني
إطبع القائمة المفروزة لما هو على اليسار
إكتب القيمة المركزية
إطبع القائمة المفروزة لما هو على اليمين .

إنها عملية تكرارية والتي نوقفها عندما نعرف أن نطبع مباشرة ما هو باقٍ للطبع : مثلاً القائمة الفارغة هي سهلة الطبع ؛ من هنا إجراء الطبع :

```
procedure editer(arbre:fils);
begin
  if arbre<>nil then begin
    editer(arbre^.gauche);
    writeln(arbre^.valeur);
    editer(arbre^.droite)
  end
end;
```

(éditer : إطبع ؛ arbre : شجر ؛ gauche : يسار ، droite : يمين ؛ valeur : قيمة) .

الذي سيتم تسميته في البرنامج بـ (racine) éditer .

لزيادة قيمة u ، نجري مقارنات متتالية حتى الحصول على مكان حر

```
y:=racine;
repeat
  x:=y;
  if u<y^.valeur then y:=y^.gauche
  else y=y^.droite
until y=nil;
if u<x^.valeur then creer(u,x^.gauche)
else creer(u,x^.droite)
```

مع الإجراء créer (الخلق)

```
procedure creer(v:chaine; var p:fils);
begin
  new(p);
  with p^ do begin
    valeur:=v; gauche:=nil; droite:=nil end
end;
```

```
program TriBinaire(input,output);
const n=20;
type chaine=packed array [1..n] of char;
     fils=^noeud;        { شجرابية ثنائية }
     noeud=record valeur:chaine;
                  gauche,droite:fils  end;
var racine:fils;
    c:chaine;

  procedure creer (v:chaine;var p:fils);
  begin new(p);
    with p^ do begin valeur:=v; gauche:=nil; droite:=nil
    end
  end;  {creer}

  procedure ajouter(u:chaine);
  var x,y:fils;
  begin y:=racine;
    repeat x:=y;
      if u<y^.valeur then y:=y^.gauche
      else y:=y^.droite
    until y=nil;
    if u<x^.valeur then creer(u,x^.gauche)
    else creer(u,x^.droite)
  end;  {ajouter}
```

```
procedure editer(arbre:fils);
begin
  if arbre<>nil then begin
    editer(arbre^.gauche);
    writeln(arbre^.valeur);
    editer(arbre^.droite)
  end
end;   { الخلق }

begin    { أضف }
  readln(c);
  creer(c,racine);
  while not eof do begin
    readln(c);
    ajouter(c)
  end;
  editer (racine)
end.
```

ملاحظة : تسمح مشتقات الفقرات والأدلاّء في الباسكال (على أكثرية الحاسوبات) ، ببلوغ فكرة العنوان : إذا كان الدليل عنواناً ويشغل كلمة ، إذا شغل عدد صحيح كلمة ، عدد حقيقي إثنتين وإذا شغلت كذلك مجموعة من 32 عنصراً كلمة ، فإذاً مع التصريح .

```
type ptr=^item;
     genre=(bits,octets,mots,entiers,reels,adresses);
     item=record
       case genre of
         bits:     (bit:packed set of 0..31);
         octets:   (octet:packed array[0..3] of 0..255);
         mots, entiers: (i:integer);
         reels:    (r:real);
         adresses: (a:ptr)
     end;
var mem:item;
```

(bits : بتات ؛ octet : بايتات ؛ mots : كلمات ؛ entiers : أعداد صحيحة ؛ réels : أعداد حقيقية ؛ adresses : عناوين) .

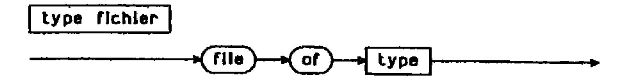
نبلّغ في الذاكرة مثلاً البتة x من الكلمة y :

بلوغ	*mem.i: = y*
إختبار	*x* **in** *mem.a↑.bit*
ضبط بـ 1	*mem.a↑.bit: = mem.a↑.bit + [x]*
ضبط بـ 0	*mem.a↑.bit: = mem.a↑.bit − [x]*

7.4 ـ سجلات (Fichiers)

تتطابق فكرة السجل مع مسلسل مركّبات ، كلها من نفس النوع ، ذوات عدد غير محدّد مسبّقاً (هذا ما يفرّقه عن الجدول) . يمكن بلوغ مركّب واحد في وقت واحد ؛ نبلُغه بواسطة نافذة نزيحها عن طريق إستعمال الإجراءات المعرّفة مسبقاً get في الشأن معاينة ، وPut في الشأن تناتج . يمكن كذلك إعادة وضع النافذة على بداية المسلسل بواسطة reset للمعاينة التسلسلية ، أو إتلاف المسلسل لنخلق منه مسلسل جديد بواسطة rewrite للتناتج التسلسلي . أخيراً عندما تصل النافذة الى نهاية المسلسل ، فإن الشرط eof يصبح صحاً .

عند تصريح السجل ، نحدد نوع المركّبات



مثال : **var *f* : file of *integer***

يمكن بلوغ المركّب الراهِن في السجل بواسطة النافذة أو المتغيّر الداريء ، المرافق للسجل :

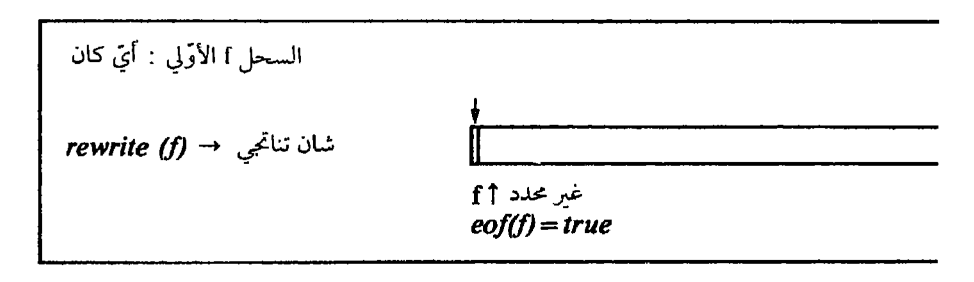


(variable tampon : متغير داريء ؛ fichier : سجل)

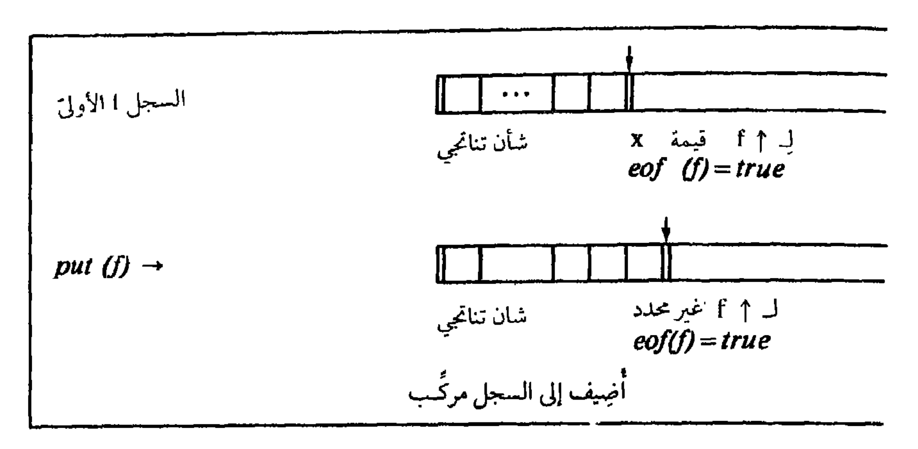
مثال : 12 = : ↑ f (لِعَدم خَلطِه مع المتغير المدلّل عليه)

rewrite (f) يضع السجل في الشأن تناتج

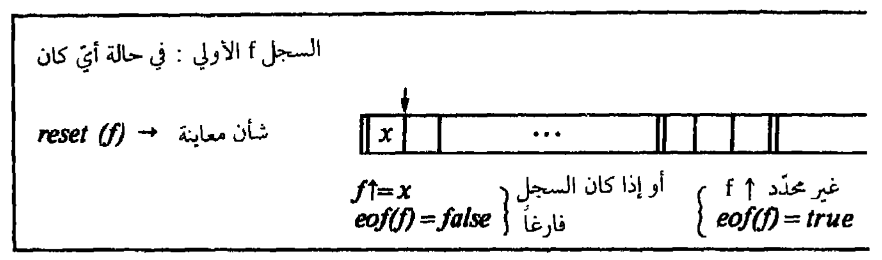
eof (f) يساوي true ، السجل فارغ (ضاعت قيمته القديمة) و ↑ f غير محدّد : يجب إعطائه قيمة قبل كل عملية put (f)



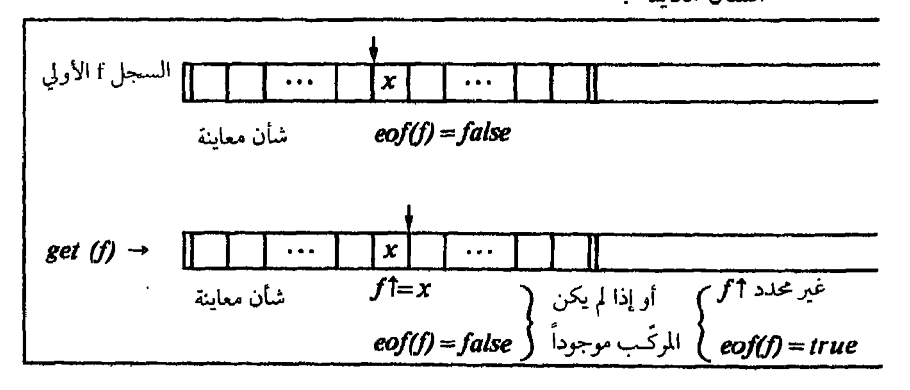
put (f) يضيف قيمة f ↑ في نهاية السجل ، الذي يجب أن يكون شأناً تناتجياً ، ويقدِّم النافذة .



reset (f) يضع السجل في الشأن معاينة
إذا لم يكن السجل فارغاً ، eof (f) تساوي false و f ↑ تساوي المركَّب الأول (تنفِّذ reset أوّل ger ضمني)



get (f) يقدّم النافذة وإذا وُجِدَ المركَّب ، عيَّنَهُ بـ f ↑ ؛ يجب أن يكون السجل في الشأن معاينة .



مثال : نَسْخ copie

إنسخ السجل f على السجل g ، السجلّين من نفس النوع

```
program copie (f,g);
{ نسخ لسجلات خارجية عن طريق استعمال get و put }
type sequence=file of record nom:integer; qte:real end;
var f,g:sequence;
   procedure copie (var u,v:sequence);
   begin
     reset(u);   {reset remplit le fenêtre u^}
     rewrite(v);
     while not eof(u) do begin
       v^:=u^;  put(v);  get(u) end
   end;
begin copie(f,g) end.
```

المعرَّفان f وg هما خارجيان عن البرنامج (أنظر 2.2) ؛ لا يمكن أن يكون السجل وسيط قيمة .

مثال تمدّد Etendre

يحضّـر هذا الإجراء ، السجل f من النوع T ، لعملية إلحاق مركّبات ،لن نضع أي إفتراض على الحالة الأوليّـة لِـ f .

```
procedure PrepareEtendre(var f:T);
var auxiliaire:T;
   procedure copie (var u,v:T);
   begin
     reset(u);   {reset remplit le fenêtre u^}
     rewrite(v);
     while not eof(u) do begin
       v^:=u^;  put(v);  get(u) end
   end;
begin copie(f,auxiliaire); copie(auxiliaire,f) end;
```

إن السجل auxiliaire (مساعد) هو مـوضعيٌّ في الإجراء : يُخلَقُ عنـد تنشيطه ويُتلَفُ عند الرجوع إلى المنادَى .

إختصارات

فيما عدا السجلات text :

1 ـ read (f, x) مُكافىء لِـ

begin x : = f ↑ ; get (f) **end**

x هو متغير يمكن تعليبُه

2 ـ read (f, x1, ..., xn) مُكافئ لِـ

begin read (f, x1) ; ... read (f, xn) **end**

3 ـ write (f, x) مُكافئ لِـ

begin f ≤7 : = x ; Put (f) **end**

x هو تعبير

4 ـ write (f, x1, ..., xn) مُكافئ لِـ

begin write (f, x1) ; ...; write (f, xn) **end**

مثال : نَسْخ

```
procedure copie2(var u,v:sequence);
var x:....;  {composant de sequence}
begin
  reset(u); rewrite(v);
  while not eof(u) do begin
    read(u,x);        {x:=u^; get(u)}
    write(v,x)        {v^:=x; put(v)}
  end
end;
```

يُعفي الشكل المختصر read / write من المعالجة المملّة للنوافذ ، لكنه يخفى عن قارىء البرنامج العمل الدقيق :

يُستعمل get بعد إستعمال النافذة

يُستعمل eof قبل إستعمال النافذة .

تتناسب إذن العبارة طالما (tant que) وبشكل جيد مع توقيف مع إقتصار :

إنتاج العنصر الأول ——————————— reset (u) ←

while شرط على العنصر المُنتج **do** ——————————— eof (u) ←

عَالِج العنصر ——————————— : = u ↑ ←

أَنتج العنصر التالي ——————————— get (u) ←

بعد تنفيذ read (u, x) ، تحتوي النافذة u ↑ على المركّب التالي المُحصَّل عليه بـ read ؛ هذا ما يشكّل فائدة كبيرة في كثير من المسائل ذات الطابع التحليلي النحوي ، التي تتطلب معرفة مسبّقة للمركّب .

مثال : فرز ـ إندماج

يتمّ الفرز في ذاكرة أساسية ، أي في جدول ، بينما تكون المعطيات في ذاكرة ثانوية ، ممثّلة غالباً بسجلٍ خارجي . غير أن الذاكرة الأساسية لها حجم محدّد ، هذا ما يحدُّ من عدد القيم التي يمكن فرزها (يتوقّف هذا العدد على الحاسوب المستعمل ، يمكن أن يكون

1000 أو 10 000 على ميكروحاسوب ، أو يتعدى المليون على حاسوب كبير) .

تتلخص الفكرة الأساس في الفرز ـ إندماج ، بتقطيع المعطيات إلى N سجل من P عنصر ، بشكل يمكّن من فرز P قيمة في الذاكرة ، N مرة متتالية ، ومن ثم دمج السجلات المفروزة .

لنفرض أننا نريد خلق سجل h ناتج عن إندماج السجلّين f وg المفروزين بالترتيب التصاعدي :

```
type fichier=file of integer;

procedure fusion(var f,g,h:fichier);
{ دمج السجلات المرتبة f و g بسجل واحد h ، هو أيضاً مرتب }

var vide:boolean;
begin
  reset(f); reset(g); rewrite(h);
  {1 دمج حتى نهاية السجل }
  vide:=eof(f) or eof(g);
  while not vide do begin
    if f^<g^ then begin
      h^:=f^;  get(f);  vide:=eof(f)  end
    else begin
      h^:=g^;  get(g);  vide:=eof(g)  end;
    put (h)
  end;  {while}
  { 2 / نسخ للنهاية المحتملة للسجل f }
  while not eof(f) do begin
    h^:=f^;  get(f);  put(h)  end;
  { 3 / نسخ للنهاية المحتملة للسجل g }
  while not eof(g) do begin
    h^:=g^;  get(g);  put (h)  end
end;  { إندماج }
```

8.4 ـ سجلّات النص (Fichiers de Texte)

إن سجل النص هو سجل سمات مركّب على هيئة سطور ؛ تُطبّق الإجراءات والدوال rewrite ، reset ، put ، get ، eof ، read ، write على معالجة السمات في سجل نص بنفس الطريقة التي تطبق فيها على سجل سمات .

يُعبّر المعرَّف المعرَّف سابقاً text عن نوع سجل النص . بفعل تركيبه على هيئة سطور ، فإن أربع عمليات إضافية هي متاحة :

1 ـ writeln (f) يضع علامة نهاية السطر في السجل f ؛ لا تكون علامة نهاية السطر ، عامة ، سمة خاصة : إنها تتوقف على الحاسوب المُستعمل ، لكن يتم

إعادة قراءتها (بـ get أو read) كسمة تباعُد .

writeln (f, e1, .. en) مُكافِئ لِـ

begin write (f, e1, ..., en) ; writeln (f) **end**

2 ـ readln (f) له الأثر بوضع الموقع الجاري للسجل f مباشرةً بعد نهاية السطر الجاري فيه العمل ؛ تجد النافذة ↑ f هكذا نفسها مركّـزة على السمة الأولى للسطر التالي في حال وجوده .

readln مُكافِئ لِـ

begin read (f, v1, ..., vn) ; readln (f) **end**

3 ـ تُرجع الدالة البولية eoln (f) القيمة true إذا كانت النافذة ↑ f مركّـزة على نهاية سطر ؛ في هذه الحالة = ↑ f ، ووحدها eoln (f) تسمح بالتفريق ما بين تباعد نهاية السطر هذا والتباعد الإعتيادي .

4 ـ page (f) يؤدي إلى طباعة النص الذي يلي على صفحة جديدة ، عندما يكون السجل f مطبوعاً على جهاز ضوئي ملائم .

إختصارات : إذا أُغفل ذكر إسم السجل ، يُطبَّـق الإجراء أو الدالة على إحدى السجلات المعرّفة مسبّـقاً input أو output :

write (output, e)	تعني	*write (e)*
writeln (output)		*writeln*
read (input, v)		*read (v)*
readln (input)		*readln*
eoln (input)		*eoln*
eof (input)		*eof*
page (output)		*page*

read وwrite يقبلان كذلك بوسائط أخرى غير النوع سمة ؛ تمّ تفصيل المعالجة في الفقرة « دَخْل ـ خَرْج » في 4.2 .

مثال : نريد إعادة كتابة نص ، مؤلف من كلمات مفصولة بتباعدات ، على عرضٍ معطى ، دون تقطيعٍ للكلمات (إلا إذا كـانت الكلمة كبيـرة جداً لتكتب عـلى سطرٍ واحد) .

1 ـ السجل i الحاوي للنص الأوّلي سيتم تصفحه كلمة بعد كلمة :

```
while not eof (i) do begin
      sauter les espaces superflus précédant un mot
      lire un mot
    s'il ne tient pas sur la ligne en cours
              aller à la ligne
  écrire le mot
end
```

Sauter les espaces superflus précédant un mot	تخطى التباعدات الزائدة السابقة لكلمة
lire un mot	إقرأ كلمه
S'il ne tient pas sur la ligne en cours	إذا لم يكفها السطر الجاري العمل فيه
aller à la ligne	إلى سطرٍ جديد
écrire le mot	اكتب الكلمة
end	

2 ـ « تخطّى التباعدات الزائدة » :

```
repeat read(i,c) until c<>' '
```

مع Var c : char

3 ـ « إقرأ كلمة » : نقوم بترتيبها في جدول كلمة (mot)

array [1... long] **of** char

حيث يكون الثابت long ، الطول الأقصى لكلمة تُكتب على سطر ، أي حجم السطر ؛ يعطي الدليل m موقع السمة الأخيرة المُدخلة في الجدول ، تساوي m إذن صفراً في البدء . تتم القراءة سمة بعد سمة ؛ الأخيرة المقروءة ، تباعداً ، لا تدخل ضمن الكلمة : تكرارية مع إقتصار ← **while**

```
b:=0;
while c<>' ' do begin              →  إستعمل
  m:=m+1;  mot[m]:=c;  read(i,c)   →  انتج
end
```

تم إنتاج الحدّ الأول c بـ « تخطى التباعدات »

4 ـ « إذا لم يكفها السطر الجاري العمل فيه ، إلى سطرٍ جديد » ، يستعين بالموقع الأول الحر على السطر ، l (في البدء 1) :

```
if (l+m-1)>long then begin  writeln(f);  l:=1  end
```

5 ـ « أكتب الكلمة »

كان السطر كافياً للكلمة ، نكتبها ، ثم نسعى لتحضير كتابة الكلمة التالية عن طريق فصلها عن الكلمة المكتوبة بواسطة تباعد .

```
for x:=1 to m do write(f,mot [x]);
l:=l+m;
if (l+1)>long then begin  writeln(f);  l:=1  end
else begin  write(f,' ');  l:=l+1  end
```

٥ ـ لكن إذا كان طول الكلمة أكبر من طول السطر ، يجب تقطيعها ، من هنا التدقيق في « إقرأ كلمة »

```
m:=0;
while c<>' ' do begin
  m:=m+1;  mot[m]:=c;  read(i,c);
  if (m=long) and (c<>' ') then begin
    {mot>ligne,le couper}
    if l<>1 then writeln(f);
    for x:=1 to long-1 do write(f,mot[x]);
    writeln(f,'-'); l:=1; mot[1]:=mot[m]; m:=1
  end
end
```

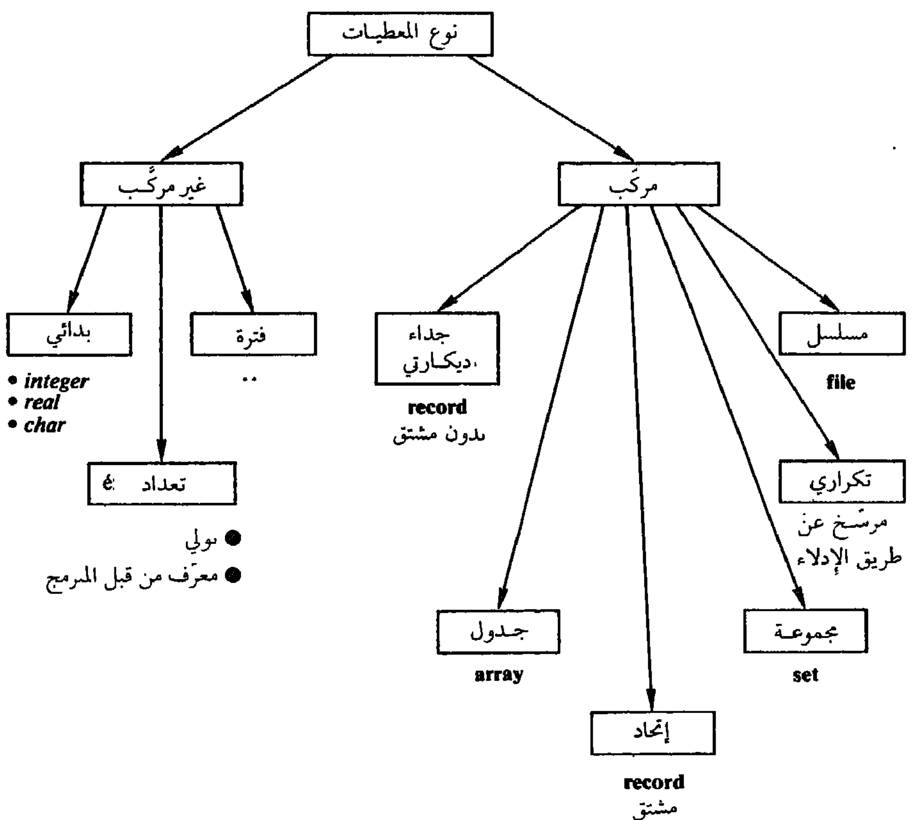
```
program reecrire(i,f);
{ أعد الكتابة على ا لنص مقروء على i مؤلف من }
{ كلمات مفصولة بتباعدات بشكل لا تتعدى طول معطي لكل سطر }
const long=36;          { طول السطر في النص النهائي }
var i,f:text;           { النص الأولي ، النص النهائي ، سجلات خارجية }
    l:integer;          { الموقع التالي الحرفي السطر }
    mot:array [1..long] of char;     { الكلمة المقروءة }
    m:integer;          { دليل في كلمة آخر سمة مُدْخلة }
    c:char;             { سمة في الدخل }
    x:integer;
begin
  reset(i); rewrite(f);
  l:=1;
  {تصفح النصّ الأولي كلمة كلمة }
  while not eof(i) do begin
    m:=0;
    { تخطى التباعدات الزائدة السابقة للكلمة }
    repeat  read(i,c)  until c<>' ';
    { أنه قراءة الكلمة }
    while c<>' ' do begin
      m:=m+1;  mot[m]:=c;  read(i,c);
      { إذا كانت الكلمة طويلة بالنسبة للسطر ، قطّعها }
      if (m=long) and (c<>' ') then begin
        if l<>1 then writeln(f);
        for x:=1 to long-1 do write(f,mot[x]);
        writeln (f,'-');  l:=1;  mot[1]:=mot[m];  m:=1
      end
```

```
      end;
      { إذا كانت الكلمة المقروءة طويلة بالنسبة لنهاية السطر الجاري العمل فيه ، إنتقل الى سطر جديد }
      if (l+m-1)>long then begin  writeln(f);  l:=1  end;
      { إكتب الكلمة }
      for x:=1 to m do write(f,mot[x]);
      l:=l+m:
     { حضّر لكتابة الكلمة التالية : ضع تباعداً لفصلها عن الكلمة السابقة }

      if (l+1)>long then begin  writeln(f);  l:=1  end
      else begin  write(f,' ');  l:=l+1  end
    end;    {while not eof (i)}
    if l>1 then writeln(f)
  end.
```

9.4 ـ التسلسل العشيري للأنواع (hiérarchie des types)

ترجع لغة الباسكال ، المعرّفة من قبل ن. ويـرث (N. Wirth) إلى فكرة البـرمجة المركّـبة ؛ على الأخصّ أنواع المعطيات الموافقة لترسيخ معنىً (مقترحة من ث . أ . ر. هوار ((C.A. R. Hoare) .



10.4 ـ متمّمات (compléments)

أعداد شبه صدفية (Nombres pseudo-aléatoires)

لا تقدِّم لغة الباسكال صراحة مولِّداً لأرقام شبه صدفية . فيما يلي بعض التقنيات اللازمة لإنجاز مولّـد :

توزيع متماثل على [1,0]

```
function alea(var germe:integer):real;
begin
  alex:=germe/65535;
  germe:=(25173*germe+13849) mood 65536
end;
```

تُكَوِّن هذه الدالة 65 536 قيمة صدفيّـة مختلفة ، موزعة بشكل متماثل على [1,0] قبل أن تتكرَّر . إنها تعمل على كل حاسوب يكون فيه $\text{maxint} \geqslant 2^{31} - 1$. يجب أن يُحفظ الوسيط المتغير germe (بذرة) من نداء إلى نداء .

لـ $\text{maxint} \geqslant z^{15} - 1$ ، يمكن إستعمال

```
function unif(inf,sup:real; var germe:integer):real;
begin
  germe:=germe*899;
  if germe<0 then germe:=germe+32767+1;
  unif:=germe/32767.0*(sup-inf)+inf
end;
```

التي تولّـد قيمتها على [أدنى ، أقصى [.

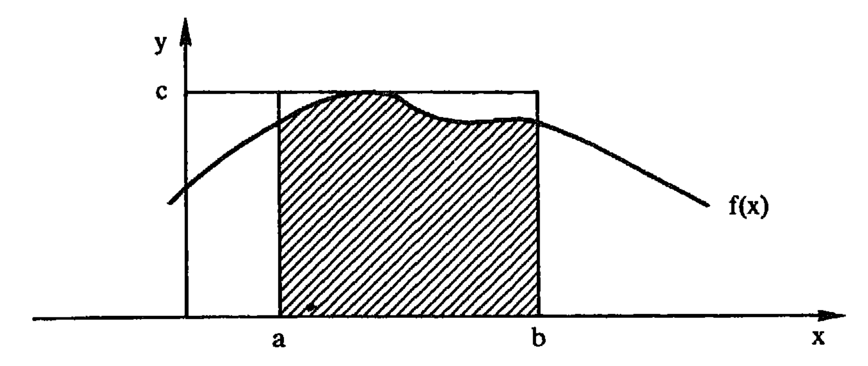
توزيع غوسي (Répartition gaussienne)

```
function Gauss(moyenne,ecartType:real;
               var germe:integer):real;
var k:integer; S:real;
begin
  S:=0;
  for k:=1 to 12 do S:=S+unif(0.0,1.0,germe);
  Gauss:=moyenne+(S-6.0)/12.0*ecartType
end;
```

(moyenne : متوسط ؛ écart type : إنحراف معياري)

مثال : مُكاملة عـلى طريقـة مونت ـ كـارلو Intégration par la méthode de Monte-Carlo.

لنفرض أننا نريد حساب مكاملة y = f(n)à على الفترة [b, a] حيث لِـ f(x) حد أقصى c :



لحساب المساحة $\int_a^b f(x)\,dx$, سنعمل على الإختيار بالقرعة لعدد كبير N من النقاط x, y في المستطيل $a \leqslant x \leqslant b$ و $0 \leqslant y \leqslant c$ ؛ إن نسبة العدد P للنقاط الواقعة ضمن المساحة $(y \leqslant f(x))$ إلى العدد الإجمالي للنقاط يقارب نسبة التكامل الى مساحة المستطيل :

$$\frac{P}{N} \xrightarrow[N \to \infty]{} \frac{\int_a^b f(x)\,dx}{(b-a)\,c}$$

```
program MonteCarlo(output);
var germe:integer;
    N,P,i:integer;  a,b,c,x,y:real;
  function rand:real;
  begin
    germe:=germe*899;
    if germe<0 then germe:=germe+32767+1;
    rand:=germe/32767.0
  end;
begin
  N:=10000; germe:=17; P:=0; a:=0.0; b:=4.0; c:=16.0;
  for i:=1 to N do begin
    x:=a+rand*(b-a);  y:=rand*c;
    if y<=sqr(x) then P:=P+1
  end;
  writeln('integrale de x*x sur [0,4]');
  writeln('valeur analytique: ',b*b*b/3.0:8:5);
  writeln('valeur approchee:  ',c*(b-a)*P/N:8:5)
end.
```

(valeur : قيمة ؛ analytique : تحليلي ؛ approchée : تقريبية)

التثلّـم Indentation

التثلم ، الذي هو فعل تصريح سطر ، ضروري بالنسبة لمقروئية البرامج :

```
if x<T[c] then
   b:=c
else
   a:=c;
```

```
if x<T[c] then
b:=c
else
a:=c;
```

نحـاول عن طريق التثلم ، تجميـع سطور البـرنامـج الموجـودة في نفس المستوى المنطقي . هذه هي حال التعريفات، التصريحات **Procedures Var, type, const, label** و**function** والعبارات المركّبة .

```
if c then E1
else E2

if c then begin
  E1;
  E2;
  ...
  En
end else begin
  Ep;
  ...
  Em
end

for i:=d to a do E

for i:=d to a do begin
  E1;
  ...
  En
end
```

```
begin
  E1;
  E2;
  ...
  En
end

while c do E

while c do begin
  E1;
  ...
  En
end;

repeat
  E1;
  ...
  En
until c
```

etc.

يتوجّـه التثلّـم للقارىء الإنساني وليس للآلة : لا يوجد نمط واحد لعرض البرامج .

البرنامج المقروء هو المثلّـم جيداً والذي يحتوي على أجزاء كثيفة بقدر كاف مفصولة بسطور بيضاء ؛ لا يجب التردُّد في وضع الملاحظات .

سهولة النقل (Portabilité)

إن البرنامج الذي يكـون عمله مرضٍ بشكـل كامـل ، غالبـاً ما يتم نقله عـلى حاسوبات أخرى غير الحاسب الذي كتب فيه . يمكن أن يكون النقل مباشراً أو شديد الصعوبة تبعاً للطريقة التي بها كُتب البرنامج . تهدف التوصيات الذاتية فقط إلى تحسين سهولة النقل هذه للبرنامج وليس لتعريف ما يكون « البرنامج الجيد » (« البرنامج الجيد » يرضي تماماً مستعمليه) .

* لا تستعمل إلا السمات المعروفة من اللغة الموحّـدة :

أرقام ، أحرف ، تباعد ، سمات خاصة :

> ( ) ↑ : ؛ , . [ ] = / * - +

{ } ` <

لكن ـ لا تختر طريقة شاذة لكتابة الأحرف (أحرف كبيرة ، أحرف كبيرة وأحرف صغيرة)
ـ لبعض السمات تمثيل متناوب :

{	}	↑	[	]
(★	★)	^	(.	.)
		@		

يجب عدم إستعماله إلّا في حال عدم صلاحية سمات الإسناد .

* لا تفترض مزايا خاصة للُعب السمات :

'a' < 'b' < ... < 'z'
'0' < '1' < ... < '9'
ord ('n') − ord ('0') = n $0 \leqslant n \leqslant 9$

مثلاً (`c`) succ أو `>` . ` + ` ليسوا كتابات سهلة النقل .

* لا تستعمل إلّا الكلمات الدليلية والمعرِّفين المعرَّفين مسبقاً في النظم (أنظر الملحق 3) ؛ بالأخص لا تستعمل مشتقات وطنية .
* لا تستعمل تمدّداً للغة حتى ولو بدا ضرورياً : لكل حالة في الباسكال تمدداتها ، غير المتساوقة مع الحالات الأخرى .
* لا تقم بإفتراضات متفائلة حول دقة الأعداد الحقيقية ، حقول الطبع الغيابية ، عدد السمات ذات المدلول لمعرِّف (إنها في بعض الأحيان 8 ، رغم النظم) ، حجم مجموعة (الإقتصار على 0...59 هو معقولاً ! ؛ النوع Set of char ليس دائماً صالحاً) .
* صرِّح في بداية البرنامج عن الثوابت لكل القيم العددية أو الأبجعددية المستعملة .
* إعزل وفسِّـر أجزاء البرنامج المتعلقة بالحاسـوب (نيل العنـاوين ، سجلات مبـاشرة

إسنادات خارجية ، خيارات التصريف ، اعداد ثمانيّة وسادس عشريّة ، إلخ ...) .

11.4 ـ تمارين

1 ـ اكتب كل التبديلات (Permutations) لكلمة من n حرف (يوجد ! n (عامليُّ n) ؛ مثلاً N = 3 SDE SED ESD EDS DSE DES) .

2 ـ لكي نكوِّد نصاً ، سنستبدل كل حرف باللاحق في الألفباء ('A' هو لاحق 'Z') ؛ اكتب إجراءات التكويد ونزع الكود .

3 ـ حوِّل عدد صحيح (1000 >) بالأحرف الكاملة .

4 ـ يساوي العدد الصحيح الكامل مجموع قاسميه ، يدخُل في ذلك 1 بينما لا يدخُلُ العدد نفسه . إحسب الأعداد الكاملة الأصغر من 500 .

5 ـ اكتب الـ n أوّل سطر من مثلث باسكال :

1	1		
1	2	1	
1	3	3	1
...			

(في السطر i ، العامود j ، عندنا C_i^j)

6 ـ لنفترض معطياً تاريخ من الماضي على شكل نهار ـ شهر ـ سنة ، احسب نهار الأسبوع الموافق . نذكِّر بأن التقويم الغريغوري حلّ مكان التقويم القيصري في العام 1582 ، وبأن سنة ألفيَّة تُقسم على 4 هي كبيسة ، ما عدا السنوات الألفيّة 00 حيث يكون رقم القرن لا يقسم على 4 .

7 ـ نريد طبع جدول كلمات موجودة في نص على أن يطبع بالنسبة لكل كلمة ، قائمة أرقام السطر الذي يحويها .

8 ـ إقرأ تعبيرا واحسب قيمته ، في لغة يكونُ المعرَّف فيها حرفاً . لا يوجد سوى النوع حقيقي والعمليات + - * / ، كذلك المزدوجات ، مع قواعد الأسبقية المعتادة . مسبَقا سنعيّن قيمة لكل متغيّر .

9 ـ إطبع جدولا للـ 100 أوّل عدد عشري من e ، قاعدة حساب اللوغاريتمات

$$\left(e = \sum_0^\infty \frac{1}{n!}\right)$$

10 ـ تلوين خارطة . لنفترض معرفة الحدود المشتركة بين عدة بلدان ، احسب اللون الذي يجب إعطاءه لكل منهم (أربعة ألوان تكفي) بشكل أن بلدين لهما حدود مشتركة لا يكنْ لهما نفس اللون .

11 ـ كوِّن متسلسلة من 100 عدد ، ثمّ في [2 ,1 ,0] بشكل أن متسلسلتين ثانويتين متجاورتين لا تكونا متشابهتين .

مثال : ... 0102102 و ... 01021010 لا يلائمان .

12 ـ ضع تماني ملكات على رقعة داما (4 × 4) بشكل أن ملكتين لا تكونا على احتكاك مباشر (تبعا لقواعد لعبة الداما) .

الفصل الخامس

مذكِّرة مساعِدة

0.5 ـ برنامج، فِدرة، مدى، منطقة Programme, Bloc, Portée, Région

0 ـ يتألف البرنامج من :

عنوان **... Program**

فدرة { تصريحات وتعريفات : (أنظر 2)
حسم **begin ... end**

نقطة

1 ـ تتبع التصريحات والتعريفات النسق الإلزامي :

Label	تصريحات الوسومات
Const	تعريف الثوابت
type	تعريفات الأنواع
var	تصريحات المتغيرات
procedure / **function**	تصريحات الإجراءات والدوال

كل قسم هو إختياري

2 ـ يُكتب العنوان :

program إسم (input, output)

لبرنامج يقوم بأعمال القراءة والكتابة .

3 ـ يتألف تصريح الإجراء (أو الدالة) من :

عنوان **.. procedure**

فدرة { تصريحات وتعريفات (أنظر 2)
حسم **begin .. end**

;

4 ـ تكونُ منطقة المعرِّف ، الفدرة التي تمّ التصريح عنه فيها ، كذلك الفدرات التي تحتويها هذه الفدرة .
5 ـ يكونُ مدى المعرِّف ، منطقته مطروح منها مناطق المعرِّفين الذين لديهم نفس كتابة الكلمات المصرَّح عنهم في فِدرات داخلية .
6 ـ لا يمكن إستعمال المعرِّف إلّا ضمن نطاق مداه .
7 ـ لدى المعرِّفين المعرَّف عنهم مسبَّقاً ، منطقة تحوي البرنامج ؛ يمكن إعادة التصريح عنهم .
8 ـ يجب التصريح عن كل معرِّف قبل إستعماله .

1.5 ـ معرِّف ، رَمْزْ ، فاصل (Identificateur, Symbole, Séparateur)

0 ـ يتألف المعرِّف فقط من أحرف وأرقام ، ويبدأ بحرف : vazy   B52 pi2
1 ـ سِماتُهُ كلها لها مدلول
2 ـ سيَّان إستعمال سمات كبيرة ، صغيرة ، أم غليظة ، إلخ . . . : aBc Abc abc هم نفس المعرِّف .
3 ـ لبعض الرموز تمثيلات متناوبة :

[	]	{	}	↑
(.	.)	(★	★)	∧ ou @

4 ـ بين كلمتين دليليَّتين معرِّفين ، ثوابت ، يجب على الأقل وجود فاصل واحد : **goto 13** هي غير سليمة .
5 ـ لا يمكن وجود فاصل داخل معرِّف ، كلمة دليليَّة أو رمز : go to هي غير سليمة .
6 ـ الفواصل هي التباعد ، نهاية السطر ، الملاحظة .
7 ـ تُكتب الملاحظة { ملاحظة } (Commentaire)
8 ـ لا يمكن إستعمال الكلمات الدليليّة (**... with, begin, var**) كمعرِّفين .

2.5 ـ متغيّرات (variables)

0 ـ إن المتغير هو كناية عن موقع في ذاكرة الحاسوب مخصَّص لإحتواء قيمة .
1 ـ تكون القيمة الأوليّة غير محددة .
2 ـ يَسمَح معرِّف لمتغير ببلوغ متغيّر .
3 ـ يجب أن تكون القيَم المعيَّنة لمتغير ، من نوع محدّد ، مُلاصق للمتغير .
4 ـ يُحيي تصريح المتغير ، متغيّراً يربط به نوعاً ومعرِّفاً : **var i : integer; x, y : char;**
5 ـ يجب التصريح عن كل معرِّف لمتغير مُستعملٍ في فدرة : داخل هذه الفدرة (متغيّر موضعي) أو في فدرة شاملة (متغير إجمالي) .
6 ـ يجب أن يتم إختيار المعرِّف بشكل يعكِسُ دور المتغير الملعوب في البرنامج :

. n, p, l وليس Nombre Davogadro, Mauvais Payeur, Longueur Donde

7 ـ يجب أن يتمّم التصريح عنه بملاحظة تحدد صراحة دوره .

8 ـ .

يمكن أن يكون المتغير	يتم وسمه إذن بـ		مثال
كامل	معرِّف	(identificateur)	i
مكوّن لجدول	معرِّف [دليل]	(identificateur) [indice]	T [i, j]
حقل فقرة	معرِّف . حقل	(identificateur, champ)	X. réel
تحريكي	دليل ↑	(pointeur ↑)	(صِلة) lien ↑
نافذة سجل	سجل ↑	(fichier ↑)	input ↑

وكل تأليف ، تبعاً individu [NoSS] . Père ↑ . date [mort] .an

للنوع : (شخص [NoSS] . أب ↑ . تاريخ [موت] . سنة)

3.5 ـ أنواع (types)

0 ـ يصف النوع القيم الممكنة لأداة وطريقة نيل .

1 ـ يربط معرِّف النوع ، معرِّفاً بنوع

type complexe = **record** réel, imaginaire: real **end;**

N = 0... maxint;

(complexe : عُمدي ، réel : حقيقي ؛ imaginaire : تخيليّ)

2 ـ إنّ تعريفات النوع هي ضرورية للتصريح عن وسائط إجراء أو دالة ، نتيجة دالة ، دليل (pointeur) ، ولتأمين تساوُق المتغيرات :

```
type T = .. Var X : T ...
    procedure (A : T) ...
A (X)
```

3 ـ لكل قيمة من نوع ترتيبي يتوافق عدد ترتيبي صحيح ؛ يتوافق ترتيب القيم مع ترتيب الأعداد الترتيبية .

4 ـ الأنواع الترتيبيّـة هي : صحيح ، سمة ، بولي ، تعداد وفترة من نوع ترتيبي .

5 ـ بالرغم من كون النوع الحقيقي غير مركّـب ، فإنه ليس نوعاً ترتيبياً .

6 ـ نطبّـق على النوع الترتيبي عمليات العلاقة = < = < > >= >
< ، الدوال succ وpred (إنتقال إلى السلف أو الخلف) ، الدالة ord (التي تعطي العدد الترتيبي) . فيما عدا السمات ، فإنه ليس لِـ ord دالّـة معاكسة .

7 ـ يكون النوع الترتيبي ضرورياً لتكوين فترة ، جدول (دليل) أو مجموعة .

8 ـ تُبنى الأنواع المركّـبة (جدول ، سلسال ، فقرة ، مجموعة ، سجل) على الأنـواع

البسيطة (حقيقي ، ترتيبي) والنوع دليل (type pointeur) .

4.5 ـ ثوابت ، اعداد حقيقية ، اعداد صحيحة ، سلاسل (constantes, réels, Entiers, Chaines)

0 ـ الثابت هو قيمة غير قابلة للتعديل من قبل البرنامج .

1 ـ يَربُط تعريف الثابت ، معرِّفاً بقيمة :

const pi = 3.14159; zéro = '0' ; n = 17 ; m n = − n;

2 ـ يُكْتَب الثابتَ الحقيقي من النوع real ، على شكل عِشْريّ (تحلّ النقطة مكان الفاصلة) : 3. 6 0.198 − 0.0 أو على شكل أسيٍّ (Ex يعني 10^x : : 3.2E1 : 0.43E−2 − 61E0 .

3 ـ يكتب الثابت الصحيح من النوع integer ، بدون جزء عشري ولا أسّ : 32 61− .
يجب أن تبقى قيمته ضمن الفترة maxint .. maxint - ، حيث يكون maxint ثابت معرَّف مسبقاً في كل حاسوب .

4 ـ يُكتب الثابت السِلسال ذي الـ n سمة من النوع **Packed array** [1..n] **of char** ، بين علامتي حذف : 'chaine' ('سلسال') .
لِكُل السمات بما فيها التباعد ، كذلك تمثيلهن الحرفي ، معنى .
الثابت السمة هو سِلسال من سمة واحدة '+' ''''
في الثابت السلسال ذي الـ n سمة ، تُكرَّر علامة الحذف :
' le fond de l''air est frais'

5 ـ يُعدّ الثابت **nil** متساوقاً مع كل الأنواع أدلاّء (pointeurs)

7 ـ يمكن إستعمال معرِّف الثابت في كلّ مكان حيث يكون الثابت واجباً :

const min = − 7; max = 7 ; **type** T = **array** min .. max **of** integer;

8 ـ وَجَب تعريف كل الثوابت في بداية البرنامج (أو الفدرة)

5.5 ـ فترة ، تعداد بولّي ، سمة ، جدول، فقرة

(Intervalle, Enumération, Booléen , caractère, Tableau, Article)

0 ـ يقيِّـد النوع فترة النوع الترتيبي لكنه يحافظ على عملياته :
(الإثنين . . . الجمعة) Lundi .. Vendredi '0' .. '9' , 7.. 18

1 ـ يعرِّف النوع تعداد قيماً مع ترتيبهن عن طريق التعداد . تكونُ القيم معرِّفين الثابت :

الإثنين الثلاثاء الأربعاء الخميس الجمعة السبت الأحد
(lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)
(inférieur, égal, supérieur) أقل يساوي أكبر

العمليّـات هي تلك الخاصة بالأنواع الترتيبية : ord, pred, succ ، علاقات

2 ـ يتوافق النوع المعرَّف مسبّـقاً بولي مع **type** boolean = (false, true) . العمليات هي العمليات المنطقية **and** (و) ، **or** (أو) ، **not** (لا) ، وعمليات الأنواع الترتيبية .

3 ـ النوع المعرَّف مسبّـقاً سمة ، **char** ، هو التعداد للسمات الممكن إستعمالها . يتم تعريفه في كل حاسوب بطريقة مختلفة ، لكنه يحتوي على الرموز الخاصة للغة ، على الأحرف ضمن تمثيل للأحرف واحد على الأقل ، مرتّـبين (..'a' < 'b') لكن ليس بالضرورة متتالين ، وعلى الأرقام ، مرتّـبين (... > '1' > '0') ومتتالين = ('0') succ) ('1' ..)

4 ـ العمليات هي تلك الخاصة بالأنواع الترتيبية : ord, pred, succ ، علاقات ، وchr مُعاكِس لـ ord (chr (n)) = nord

5 ـ النوع جدول هو مجموعة متغيرات كلّـها من نفس النوع :

array [دليل] **of** مكوِّن

حيث أن الدليل هو نوعاً ترتيبياً والمكوِّنات من نوع أيّ كان :

array [0..7] **of** char

اختصار : **array** [boolean ، '0'.. '7'] of reel مكافىء لِـ

array [boolean] **of array** ['0' .. '7'] **of** real

6 ـ في متغير جدول ، يكون الدليل (indice) كناية عن تعبير يُكتب بين معقّفين : T [i]

إختصار : x [false, '2'] مكافىء لِـ x[false] ['2']

الكتابة x [b] تعني سطراً ؛ لا يوجد تنويط للأعمدة .

7 ـ النوع فقرة هو مجموعة متغيرات ، كلّ منها من نوع أيّ كان :

record c1; T1; c2, c3 : T2 **end**

يستعمل المشتق منقاة للنيل :

record a : T1; **case b**: boolean, **of** true : (i: integer) ; false: (r: real) **end**

8 ـ في المتغير حقل فقرة ، يكون إسم الحقل مسبوقاً بنقطة : Y.c1

6.5 ـ تعيين (Affectation)

0 ـ تسمح عبارة التعيين بتعيين قيمة تعبير ، لمتغير من نوع متساوق :

variable : : = expression

مثال : i : = i + 1 x : = pi + 1.0

1 ـ يمكن تعيين عدد صحيح لعدد حقيقي ، إن هذا هو حال التغيير الأوتوماتي الوحيد .

2 ـ لكي نعيّن عدد حقيقي لعدد صحيح ، نستعمل القطع (أو البتر) (trunc) أو التكبير (التدوير) (round)

3 ـ يكونُ التعبير مؤلّفاً من تعبير بسيط ، أو من تعبيرات بسيطة ومؤثر علاقة (= <
(in < >   >=   > < =

x ❋ y + z / y   x − y = z + y ❋ y

4 ـ يتكوّن التعبير البسيط من متأثرات ومؤثرات ضمن إطار ترتيب الأسبقيّات التنازلية :

not
❋ / **and div mod** ضَرْبي
+ − **or** جَمْعي

5 ـ ضمن الأسبقية المتساوية ، يتم التقييم من اليسار إلى اليمين :
x ❋ y + y / z = not q تعني
(((x ❋ y) + (y/z)) = (not q))
يسمح وضع الأقواس دائماً بفرض ترتيب الحسابات .

6 ـ + − ❋ div mod عند تطبيقهم على أعداد صحيحة ، يجعلون النتيجة صحيحة ، مثل الدوال المعرَّفة مسبقاً succ, pred, ord, sqr, abs .
trunc وround يقومان بالتحويل من الحقيقي إلى الصحيح ؛ odd (x) هي صحّ إذا كان x مفرداً ؛ chr (x) تحوّل الصحيح إلى سمة .

7 ـ + − ❋ عند تطبيقهم على أعداد حقيقية ، يجعلون النتيجة حقيقية ، مثل الدوال المعرّفة مسبقاً abs وsqr . / هو دائماً قسمة حقيقية ؛ sin ، cos ، exp ، ln ، sqrt ، arctan هنّ دوال ذوات نتيجة حقيقية .

8 ـ يمكن أن يكون المتغير المعيّن من كل نوع ما عدا السجل : بسيط ، دليل (pointeur) جدول ، فقرة ، مجموعة ، سلسال ، . . .

7.5 ـ إذا ، الحالة ، لـ ، طالما ، كرِّر ، مع (si, cas, pour, tant que, répéter, Avec)
إقرأ ، أكتب READ, write

0 ـ الشرط هو تعبير يُعطي تقييمه إما القيمة صح (true) ، إمّا القيمة خطأ (false)
1 ـ في العبارة إذا : **if** condition **then** énoncé **else** énoncé
أو : **if** condition **then** énoncé

لا تُنفَّذ العبارة **then** إلاّ إذا كان الشرط صحّاً ، بينما لا تُنفَّذ العبارة **else** إلا إذا كان

الشرط خطأً . يمكن أن تكون العبارات بسيطة ،) إذا ، تعيين ، لِـ ، . . .) أو مركّبة (**begin** énoncé; énoncé ... **end**)

مثال :

if min > T [a] **then** min : = T[a]

2 ـ في العبارة الحالة

case expression **of** c1 : E1 ; c2: E2; ...; cn : En **end**

التعبير ، من نوع ترتيبي ، عليه أن يأخذ قيمة إحدى الثوابت ci ، تُنفَّذ عندها العبارة Ei وهي وحدها .

مثال :

Case comparaison (x, T [c]) **of** inférieur: b : = c ; égal : y : = c ; supérieur : a : = **c end**

3 ـ العبارة لِـ

for variable : = expression 1 expression 2 énoncé

تسمح بتكرار العبارة ، لكل قيم المتغير الذي هو من نوع ترتيبي : عبارة 1 ، succ (عبارة 1) ، succ (succ . . . عبارة 2) . إذا كان التعبير 2 < التعبير 1 ، فإن العبارة لا تُنفَّذ . إنّ إستبدال **to** بـ **downto** ، يحمل على تطبيق **pred** بدلاً من succ . بعد التنفيذ ، يكون للمتغير قيمة غير محددة .

مثال :

S : = 0 ; **for** a : = 0 **to** N- 1 **do** S : = S + Z [a]

4 ـ في العبارة طالما

while condition **do** énoncé

طالما الشرط صحاً ، فإن العبارة تُنفَّذ .

مثال :

f : = 1 ; i : = 0; **while** f < 100 **do begin** i : = i + 1 ; f : = f * i **end**

5 ـ في العبارة كرِّر

repeat énoncés **until** condition

تُنفَّذ قائمة العبارات حتى يصبح الشرط صحاً .

مثال :

repeat read (c); x: = succ (x) **until** c = ' '

6 ـ تُجمِّـع العبارة مع ، منافذ (نيل) ضمن الفقرة .
مثال :

with p ↑ [i] do a : = b − t

يتوافق مع p ↑ [i] . a : = p ↑ [i] . b − p ↑ [i] . t

7 ـ read (a, b) تقرأ قيمتين على سجل الدّخْل input وتعيّـنهما للمتغيرات a وb يمكن قراءة أعداد صحيحة ، حقيقية أو سمات .

8 ـ write (e) تكتب قيمـة التعبير e عـلى سجل الخَـرْج output . يمكن كتابـة اعـداد صحيحة ، حقيقية ، سمات ، سلاسل ، بولي لكن ليس جدولاً أو فقرة أو مجموعة أو دليلاً (pointeur) . يمكن تحديد حقل الطباعة :

write (car: 3, entier : N, réel : total : dec)

writeln تُنهي السطر الجاري العمل فيه على output .

8.5 ـ إجراء ، دالّـة ، مجموعة ، سلسال ، سجل ، دليل

(Procédure, Fonction, Ensemble, Chaîne, Fichier, Pointeur)

0 ـ يُعطي تصريح الإجراء إسماً لعبارة مركّبة. يُمكن ضمن إجراء، التصريح عن متغيرات، أنواع ، . . . (هذه فِدرة) وإعطاء إسـماً لأدوات لا تكُون فعليـاً معروفـة إلاّ عند النداء : وسائط صورية .

مثال :

```
procedure P (i, j : integer); var l, c : integer;
begin for l: = 1 to i do begin
        for c: = 1 to j do write ('★'); writeln end
end;
```

1 ـ إن عبارة نداء الإجراء تُنشِّط الإجراء وتحدِّد الوسائط الفعليَّـة ؛ مثلاً : P(15, 10) يمكن أن يكون **الوسيط** قيمةً (قيمة الـوسيط الفعلي أُعْـطِيَتْ إلى الوسيط الصُـوَري) ، متغيّـراً (الوسيط الصوَري يعطي منفذاً إلى الـوسيط الفعلي) ، إجـراءاً ، دالةً أو جدولاً ضبيطاً .

مثال :

```
procedure sigma (a, b : real; var c : real); begin c: = a + b end;
alors sigma (y ★ z/2.0, sqr (i2), x)
a pour effet x: = y ★ z/2.0 + sqr (i2)
```

فإذن لِـ
المفعول

2 ـ تتصرّف الدالّـة كالإجراء ، لكن تعطي نتيجة :

مثال :

```
function S (x, y : real) : real begin S: = x + y end;
x: = S (y ★ z/2.0, sqr (i2))
```

3 ـ النوع مجموعة هو مجموعة قيم من نوع ترتيبي . تُطبَّق على المجموعات عمليات الإتــحاد (+) ، التقاطع (*) ، الفرْق (−) ، المساواة (= و <>) ، التضمين (<= و >=) والإنتماء (in) .

يقـوم مُنشيء المجموعـات [. . .] بتمريـر قيم ترتيبيَّـة إلى المجمـوعـة ؛ تنـوَّط المجموعة فراغ بـ [ ] .

مثال :

```
var possible set of (noir, jaune, rouge, vert); C : set of char;
    if noir in possible then ...
    C: = ['0'..'9', '.', 'E']; if lu in C then ...
```

4 ـ سلسال من n سمة هو من النوع

Packed array [1.. n] **of** char

يمكن كتابته ، تعيينه ، مقارنته .

مثال :

software : = 'logiciel' avec **var** software: **packed array** [1..8] **of** char

5 ـ النوع سجل هو تتالي مركَّبات ، كلها من نفس النوع ، والتي تمُرُّ عليها نافذة بشكل متتالٍ ؛ فقط المركَّب المكشوف من قبل النافذة يمكن بلوغه .

تصريح :

f : **file of** type du composant

النافذة هي متغيِّـر ننوِّطه f ↑ .

العمليات :

get (f) إقرأ — reset (f) من جديد خُذْ موضع البدء

Put (f) أُكتب — rewrite (f) أتلِف ، لكي تَكتب

eof (f) إختبِر نهاية السجل

مثال : نسخ السجلات

```
reset (f); rewrite (g); while not eof (f) do begin
    get (f) ; g ↑ : = f ↑ ; put (g)
end
```

6 ـ النوع المعرَّف مسبقاً text هو سجل سمات مركَّب على هيئة سطور . تُركَّز نهاية السطر (يحدَّد موقعها) ، كتابةً بـ writeln (f) ؛ بينما تُختبر ، قراءةً ، بـ eoln (f) ويُمكن كذلك قراءة أو كتابة أدوات من نوع غير السمات (سيتمُّ تحويلهم إلى سلسال سمات) : صحيح ، حقيقي . . .

7 ـ يقدِّم الدليل (pointeur) منفذاً إلى متغير تحريكي من نوع محدَّد .

مثال :

```
    type ptr = ↑item; item = record val : real; lien : ptr end
    var p, tête : ptr;
p : = tête; while p < > nil do begin write (p↑.val); p: = p↑.lien end
```

8 ـ يُخلقُ المتغير التحريكي بواسطة new ويُتلَف بواسطة dispose ؛ لا تخضع مدة حياته إلى قواعد الفدرات .

مثال :

```
new (p); p↑.val:= x; p↑.lien:= tête; tête:= p;
```

الفصل السادس

ملحقات

ملحق 0 : دليل البرامج

ملحق 1 ـ مظاهر داخلية

لا يمكن لحاسوب ، إلّا تنفيذ التعليمات المكتوبة في لغة خاصة به : لغة الآلة (تُسمّى في بعض الأحيان بكلمة في غير محلّها : مؤوّل) . عبارة الباسكال ليست مباشرة قابلة للتنفيذ ، يجب في البدء ترجمتها إلى تعليمات في لغة الآلة ، إمّا مباشرة بواسطة **مصرِّف** ، إمّا بطريقة غير مباشرة بواسطة **مُفَسِّر** (p ـ كود ، UCSD . . .) ، أقل بطأً من المصرِّف .

تبقى التعليمات والمعطيات في **الذاكرة** ، وحدة النيل في الذاكرة هي **الكلمة** ، المؤلفة من عدد ثابت من **البتات** ، كل بتة ، أو موقع ثنائي يساوي إمّا 1 إمّا 0 (صفر) ؛ يمكن أن يوجد من 4 إلى 128 بتة في الكلمة (عادة 8 ، 16 أو 32 على الميكروحاسوب ، 16 أو 32 على الميني حاسوب ، 32 ، 48 ، 60 أو 64 على الحاسوبات الكبيرة) . تَشْغَل **البايتة** 8 بتة . يمكن نيْل كل كلمة فقط بواسطة **عنوانها** .

يمكن أن تتوافق أنواع المعطيات في الباسكال مثلاً (النظم لا تفرض تمثيلاً موحداً) :

صحيح : على كلمة مكوّدة في القاعدة 2 (مع إصطلاح خاص للأعداد الصحيحة السالبة) . على كلمة من n بتّة ، يمكن أن تأخذ الأعداد الصحيحة 2^n قيمة ، من -2^{n-1} إلى $2^{n-1} - 1$ ، هذا ما يحدّد قيمة maxint .

حقيقي : على كلمة ، اثنتين أو أربع كلمات تبعاً للحاسوب ، تُمثِّل كلٌ على حدة الأسّ والجزء العشري : مثلاً بـ 32 بتة ، يأخذ الأسّ قيمة تصل إلى 38 والجزء العشري من 6 إلى 7 أرقام عشرية .

بولي : على بتة (في كلمة ، تكون البتات الأخرى غير مستعملة)

جدول : على n × p كلمة في حال وجود n مركّب كلٍّ من p كلمة ؛ يتم النيل من مركّب بالتقسيم (indexation) : القيمة المحسوبة للإزاحة من الكلمات في الجدول (دليل) تُضاف إلى عنوان أوّل مركّب .

فقرة : مثل حال الجدول ، لكن يكون الدليل قيمة ثابتة (متوافقة مع إسم الحقل المُنال) ، وليس محسوبة .

رصّ : **array** [1.10] **of** boolean تَشغل 10 كلمات ، لكن تُستعمل فقط بتة واحدة بالكلمة ؛ تَشْغَل **Packed array** [1.10] **of** boolean عشر بتات متتالية .

مجموعة : متسلسلة بتات متتالية ؛ العمليات على المجموعة (+ − * in) تكون إذن عمليات على البتات . إذا سمح به حجم الكلمة ، فإن حجم المجموعة محدد

(مثلاً مع كلمات من 60 بتة ، مجموعات مركّبة على 0...59) هذا ما يمنع النوع
set of char ...

سمة : غالباً على بايتة (إصطلاح : نشطب الأصفار) .

لُعب سمات (CDC) Display Code Pascal : 6 بتات

d \ u	0	1	2	3	4	5	6	7	8	9
0		A	B	C	D	E	F	G	H	I
1	J	K	L	M	N	O	P	Q	R	S
2	T	U	V	W	X	Y	Z	0	1	2
3	3	4	5	6	7	8	9	+	−	☆
4	/	(	)	$	=		,	.	'	[
5	]	:	≠	{	v	∧	↑	}	<	>
6	≤	≥	¬	;						

عدد ترتيبي = u + 10 * d

مثال : ord ('A') = 1   chr (45) = ' ' (تباعد)

الأعداد الترتيبيّـة الأكبر من 63 والصفر ليست مُنتقاة ، الأحرف متتالية ، لا يوجد أحرف صغيرة .

لعب سمات ASCII بدون شفعية (Parité) : 7 بتات

y \ x	0	1	2	3	4	5	6	7
0	nul	dle		Ø	@	P	`	p
1	soh	dc1	!	1	A	Q	a	q
2	stx	dc2	"	2	B	R	b	r
3	ext	dc3	#	3	C	S	c	s
4	eot	dc4	$	4	D	T	d	t
5	enq	nak	%	5	E	U	e	u
6	ack	syn	&	6	F	V	f	v
7	bel	etb	'	7	G	W	g	w
8	bs	can	(	8	H	X	h	x
9	ht	em	)	9	I	Y	i	y
10	lf	sub	★	:	J	Z	j	z
11	vt	esc	+	;	K	[	k	{
12	ff	fs	,	<	L	\	1	|
13	cr	gs	-	=	M	]	m	}
14	so	rs	.	>	N	↑	n	~
15	si	us	/	?	O	_	o	del

عدد ترتيبي = 16 * x + y

مثال 65 = ord ('A')   ' ' chr (32) = O (تباعد)

الأحرف متتالية ؛ تميّز بين الأحرف الصغيرة والكبيرة . تكون سمات العدد الترتيبي من 0 إلى 31 ، و127 غير قابلة للطباعة ، إنها سمات تحكّم مُستعملة لإرسال المعطيات (cr = عودة (return) ، bel = إنذار (alarme) ، bs = عودة إلى الوراء ,... retour arriere) .

لعب سمات EBCDIC 8 0 بتة

ضمن السمات القابلة للطبع ، نجد

		espace	64	a	129	n	149	A	193	N	213
Ø	240	.	75	b	130	o	150	B	194	O	214
1	241	(	77	c	131	p	151	C	195	P	215
2	242	+	78	d	132	q	152	D	196	Q	216
3	243	☆	92	e	133	r	153	E	197	R	217
4	244	)	93	f	134	s	162	F	198	S	228
5	245	;	94	g	135	t	163	G	199	T	227
6	246	/	97	h	136	u	164	H	200	U	228
7	247	,	107	i	137	v	165	I	201	V	229
8	248	–	110	j	145	w	166	J	209	W	230
9	249	:	122	k	146	x	167	K	210	X	231
		,	125	l	147	y	168	L	211	Y	232
		=	126	m	148	z	169	M	212	Z	233

succ ('i') ≠ 'j'   *succ ('r') ≠ 's'*

(espace · تباعد)

مكدس وكُدْس (pile et tas)

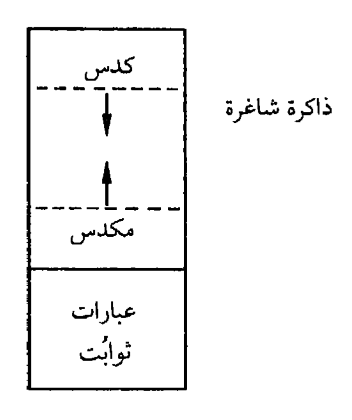
عند بداية التنفيذ ، تَنْوَجد فقط المتغيرات المصرَّحة في مستوى البرنامج ، إذن الإجماليّة لكل الإجراءات والدوال ، وكذلك السجلات input وoutput . عند تنشيط إجراء ، أو دالّة تُخلق متغيراتهن الموضعية ؛ يتم فيها بعد إتلافهن عند نهاية التنشيط (عند « العودة إلى المنادي ») : تكون فِدرات الذاكرة مُكدَّسة ، ومن ثم مزالة ، ضمن منطقة في الذاكرة تُسمّى مكدس . في كل لحظة ، تكون بذلك المتغيّرات الموضعيّة المُمكن بلوغها في الفدرة في قمة المكدس ؛ تُحدَّد بذلك مدة حياتهم بمدّة تنشيط الإجراء ، أو الدالّة ، هذه التقنية بتكرار النداءات .

ملاحظة : يُعْرَفْ حجم كل فدرة من المتغيرات الموضعية قبل تنفيذ البرنامج ، هذا

ما يسمح بإدارة النداءات الفعالة خاصة وهذا ما يستتبع تنفيذ سريع .

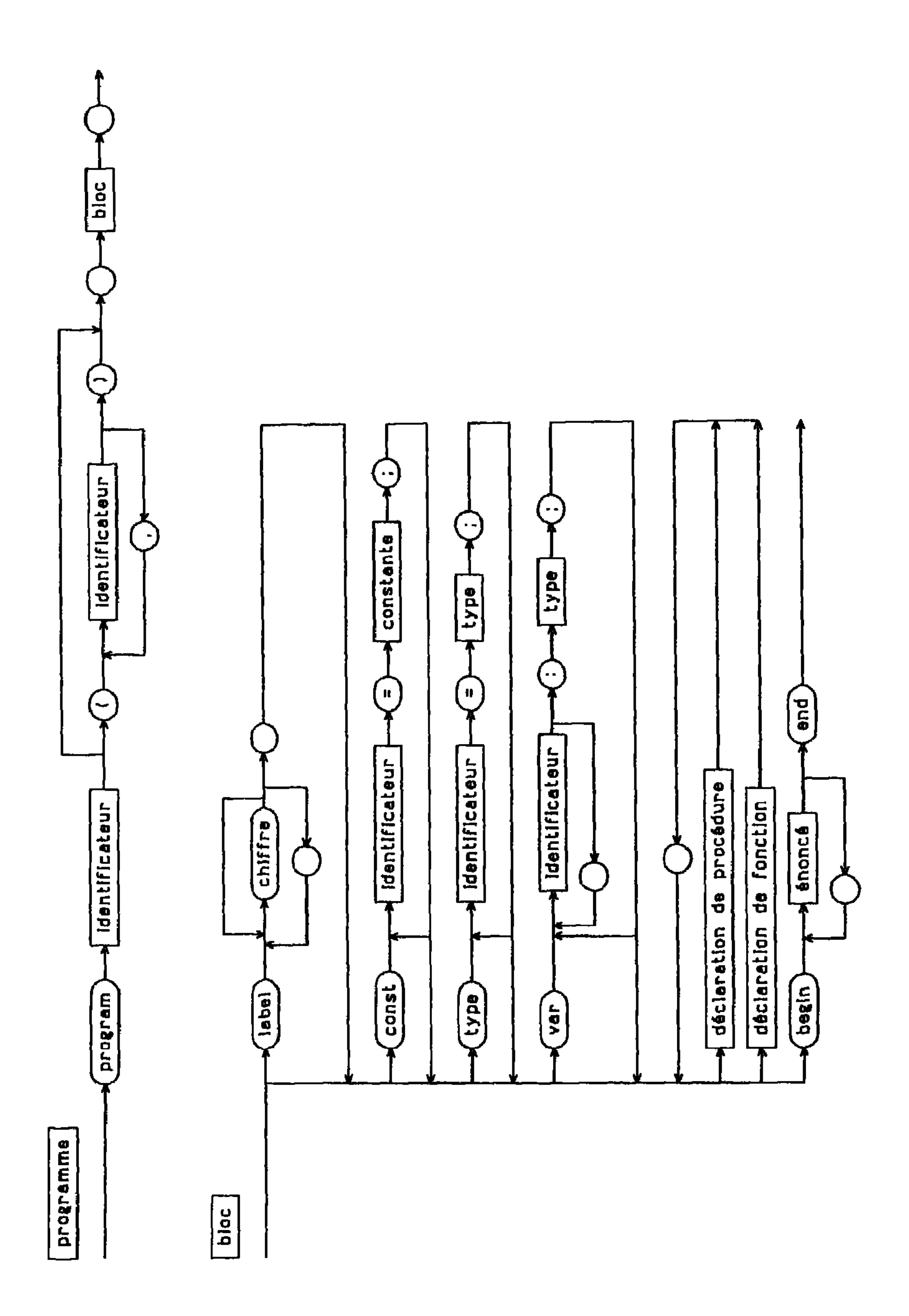
في المقابل ، يكون للمتغيرات التحريكية (المخلوقة بـ new) مدة حياة تمتد من لحظة خلقهن (new) حتى نهاية تنفيذ البرنامج ، أو حتى إتلافهن المتعمّد (بواسطة dispose) : لا يمكنهن البقاء في المكدس . يتم ترتيب هذه المتغيرات التحريكية ضمن منطقة في الذاكرة تسمّى **كُدْس** مميّزة عن المكدس .

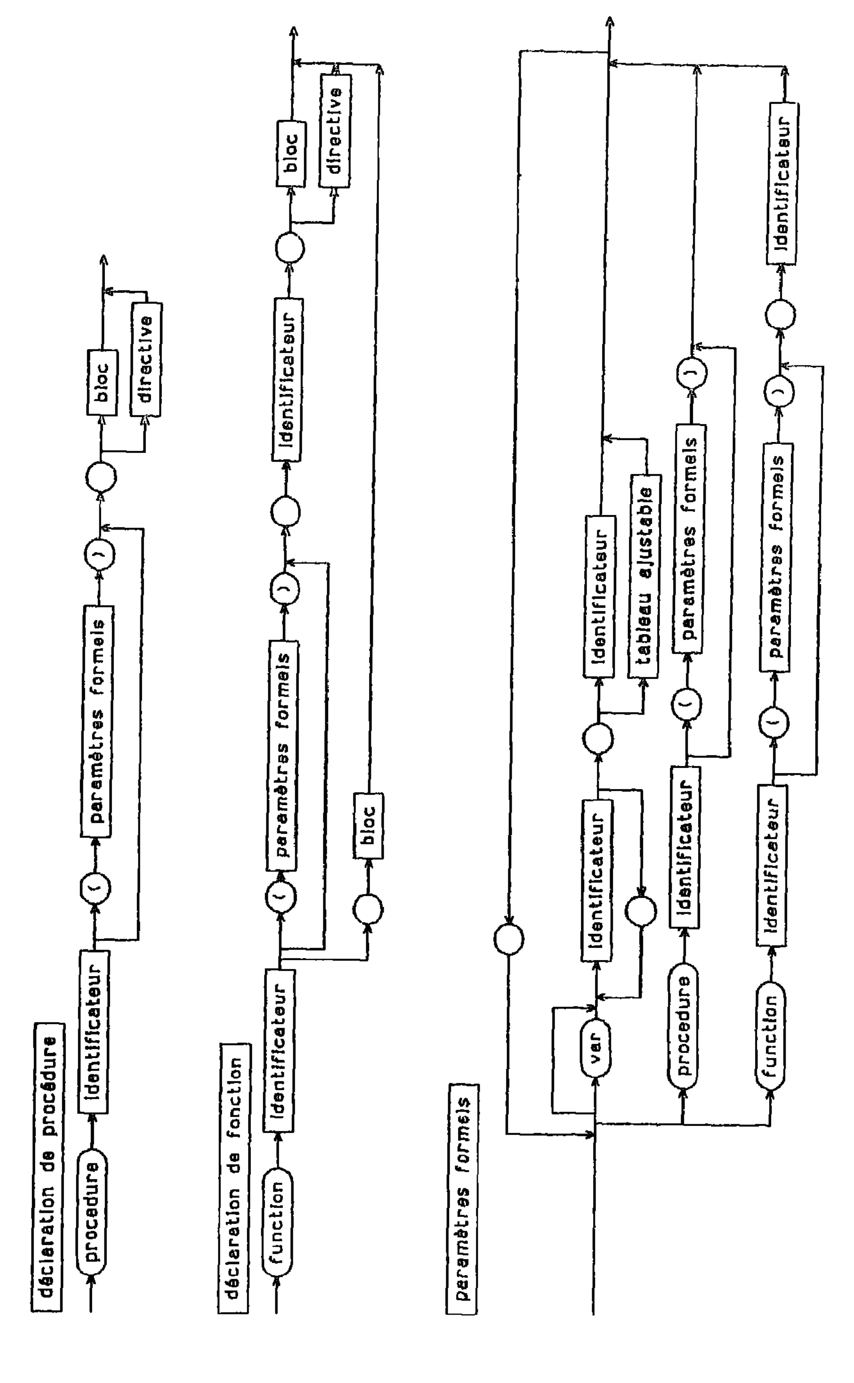
ليس من علاقة ما بين التطور لكلّ من المكدس والكدس . لكي نؤمّن إشغال أفضل للذاكرة الشاغرة ، نفضل عدم الإشغال الثابت للكدس والمكدس (يمكن أن يكون المكدس مشبعاً بينما يكون الكدس فارغاً !) ، بل بالأحرى العمل على توسيعهما كدس ومكدس باتجاه متعاكس :

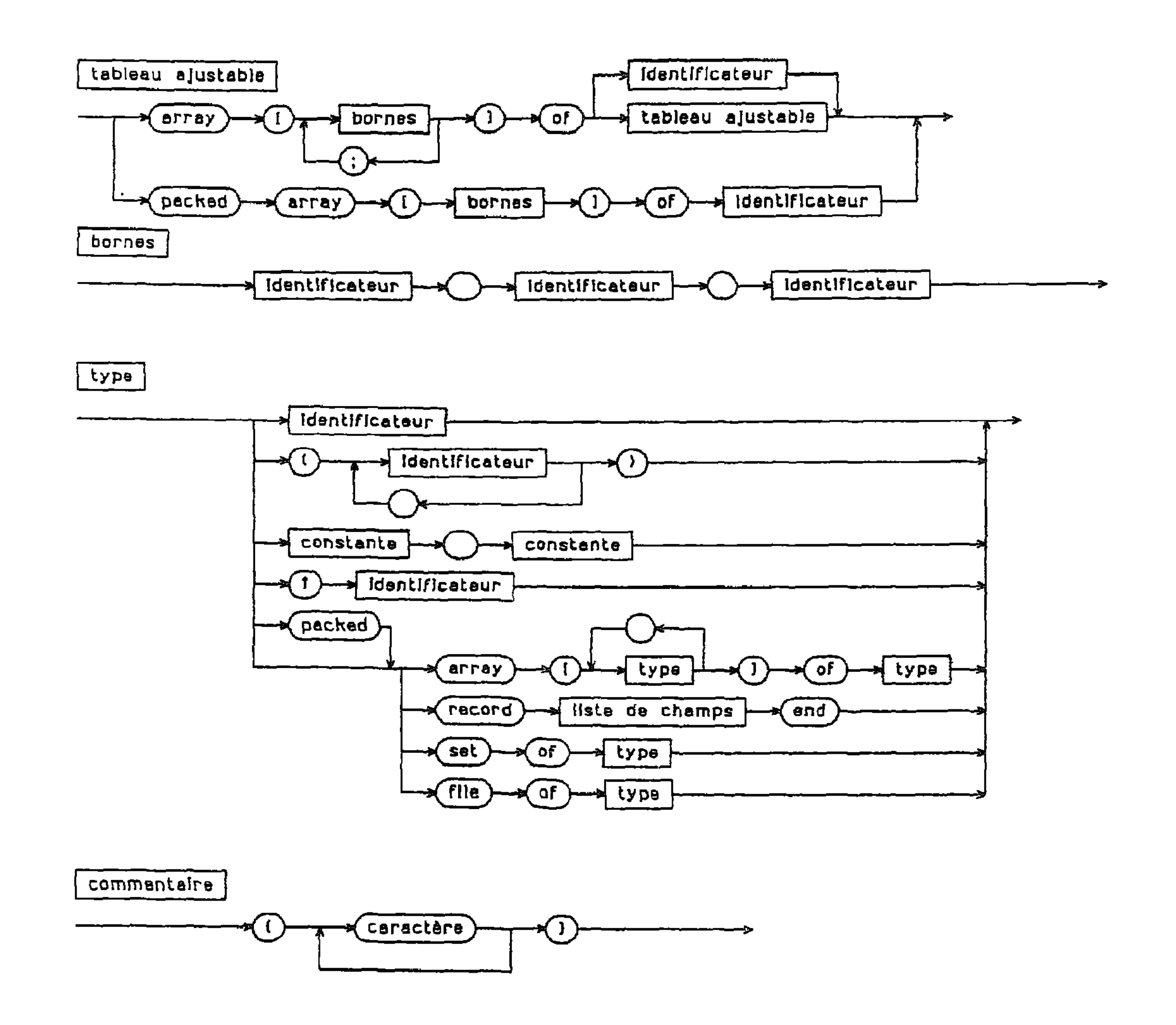


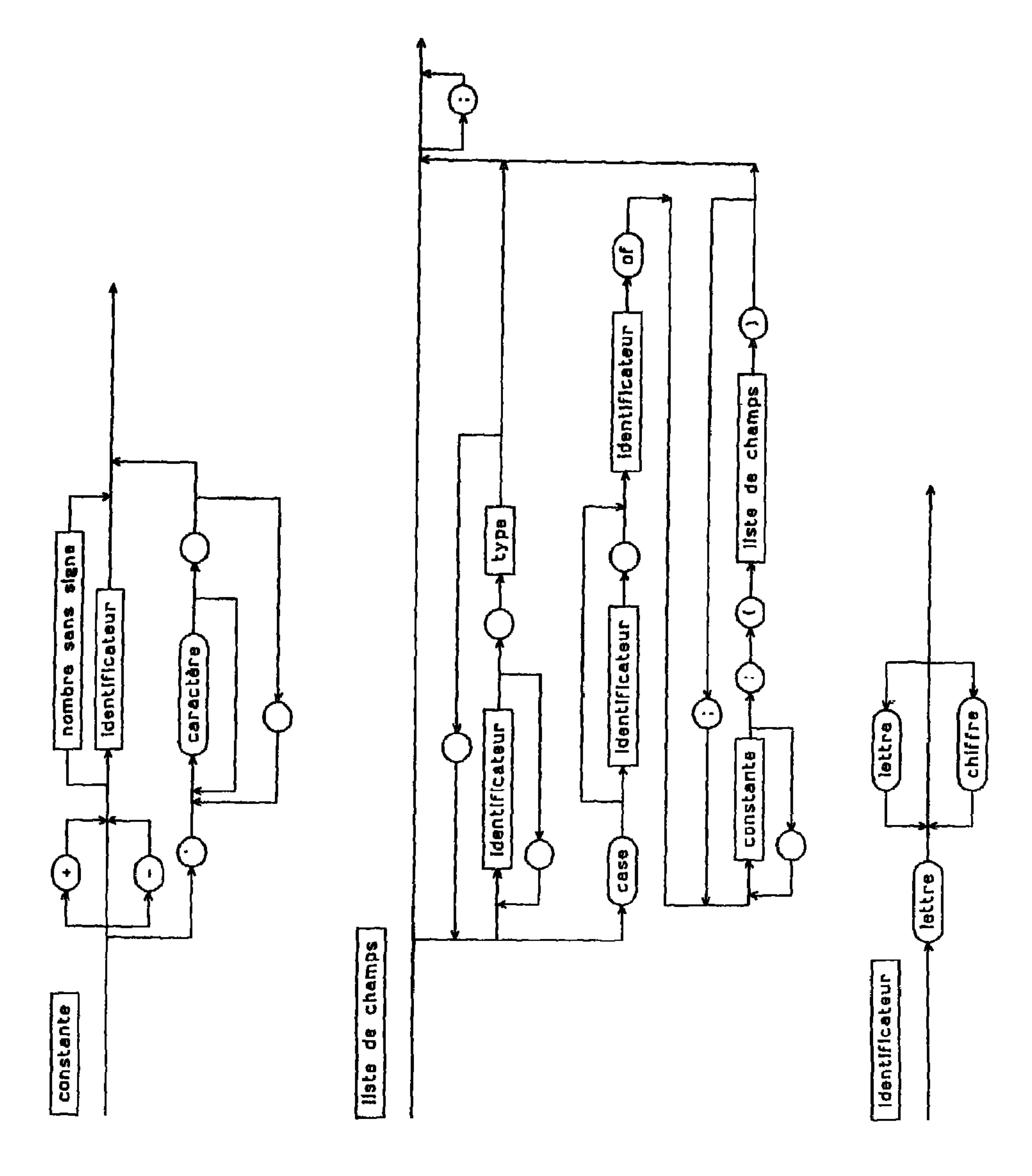
يتوقف التنفيذ إذا ما إلتقى الكدس بالمكدس لأنه يكون قد غدت حينها كل الذاكرة الشاغرة ، مُشبَعة .

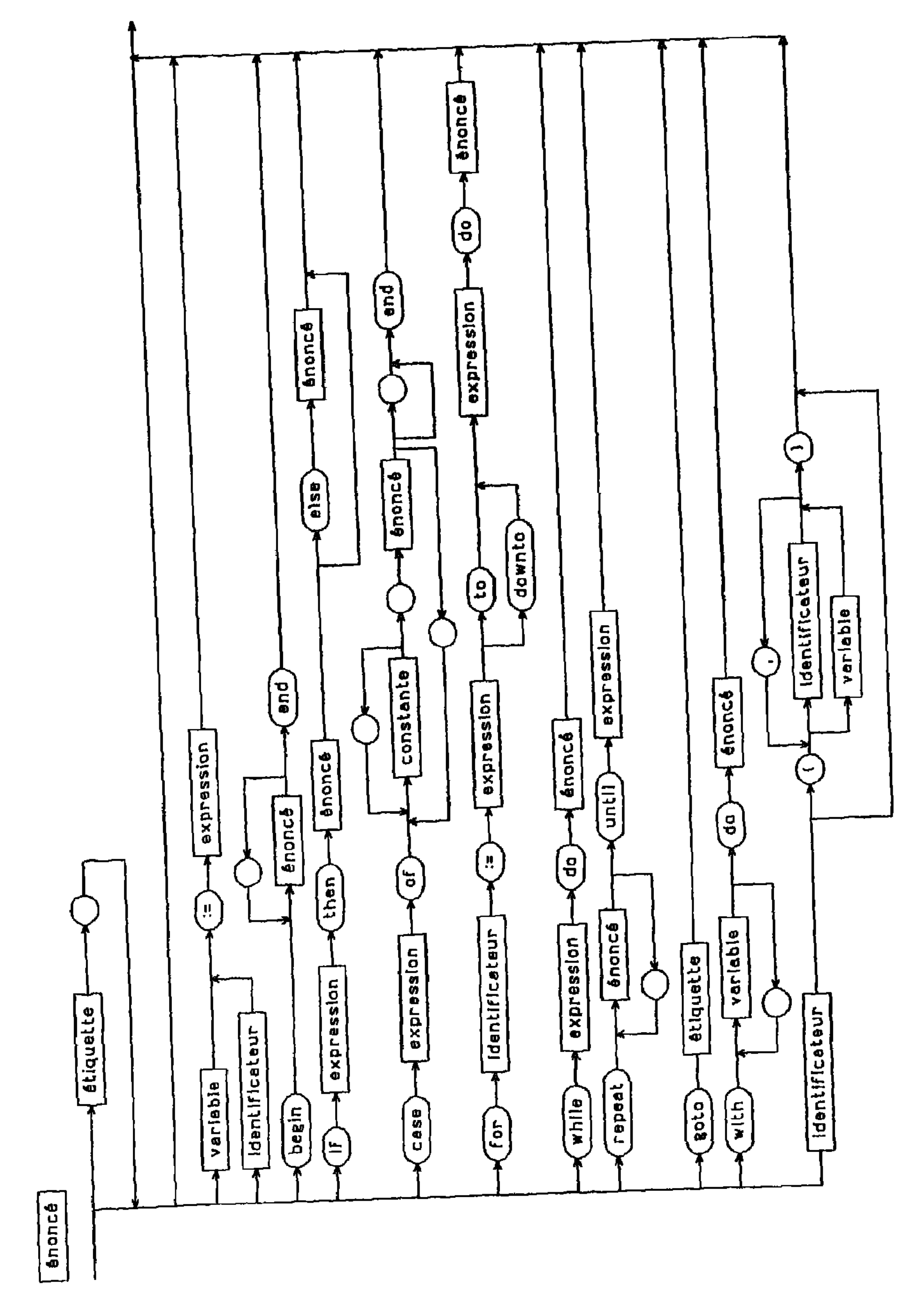
ملحق 2 : مخططات النحو



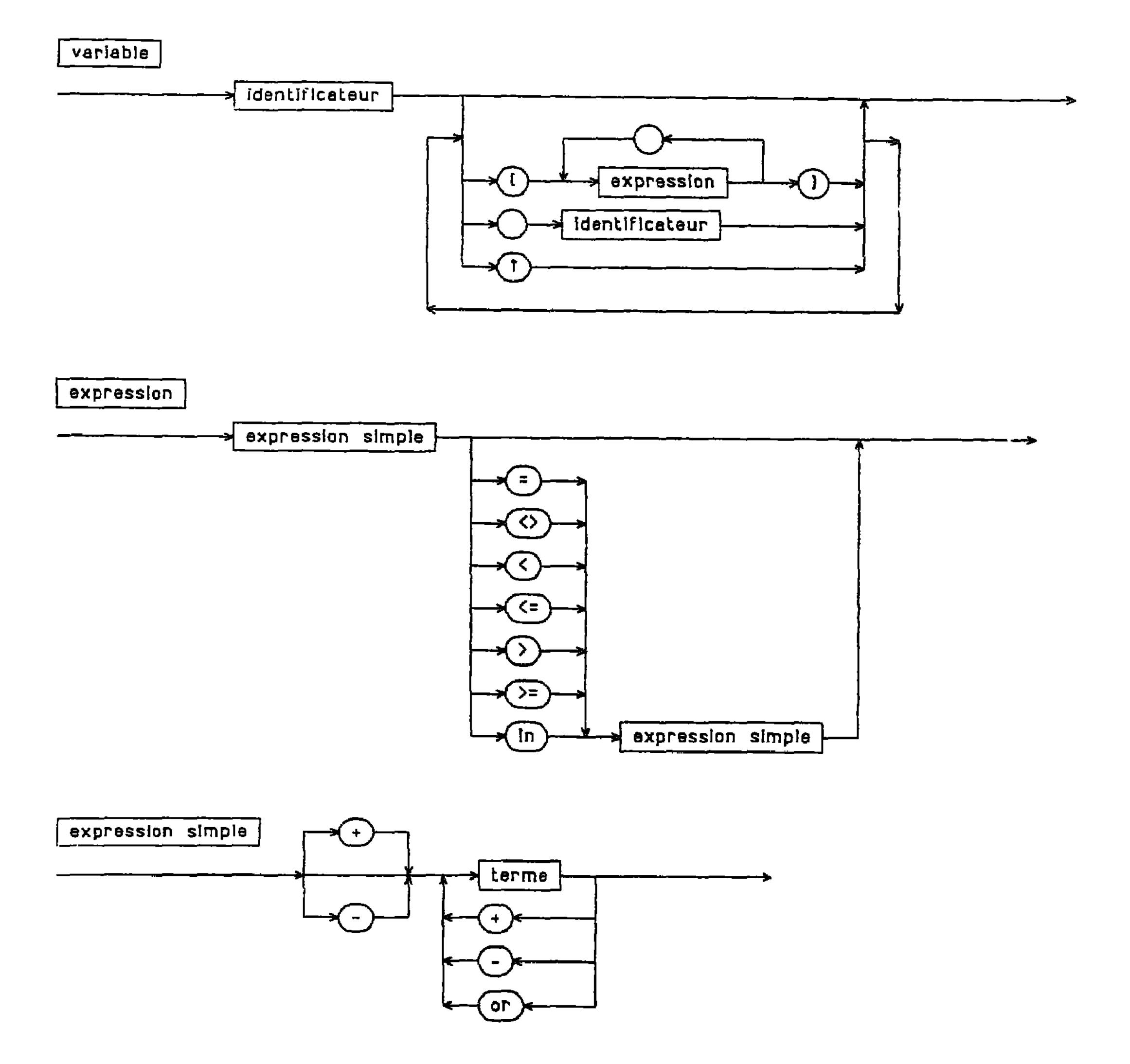

déclaration de procédure
procedure
identificateur
(
paramètres formels
)
bloc
directive
déclaration de fonction
function
identificateur
(
paramètres formels
)
identificateur
bloc
directive
bloc
paramètres formels
var
identificateur
identificateur
tableau ajustable
procedure
identificateur
(
paramètres formels
)
function
identificateur
(
paramètres formels
)
identificateur

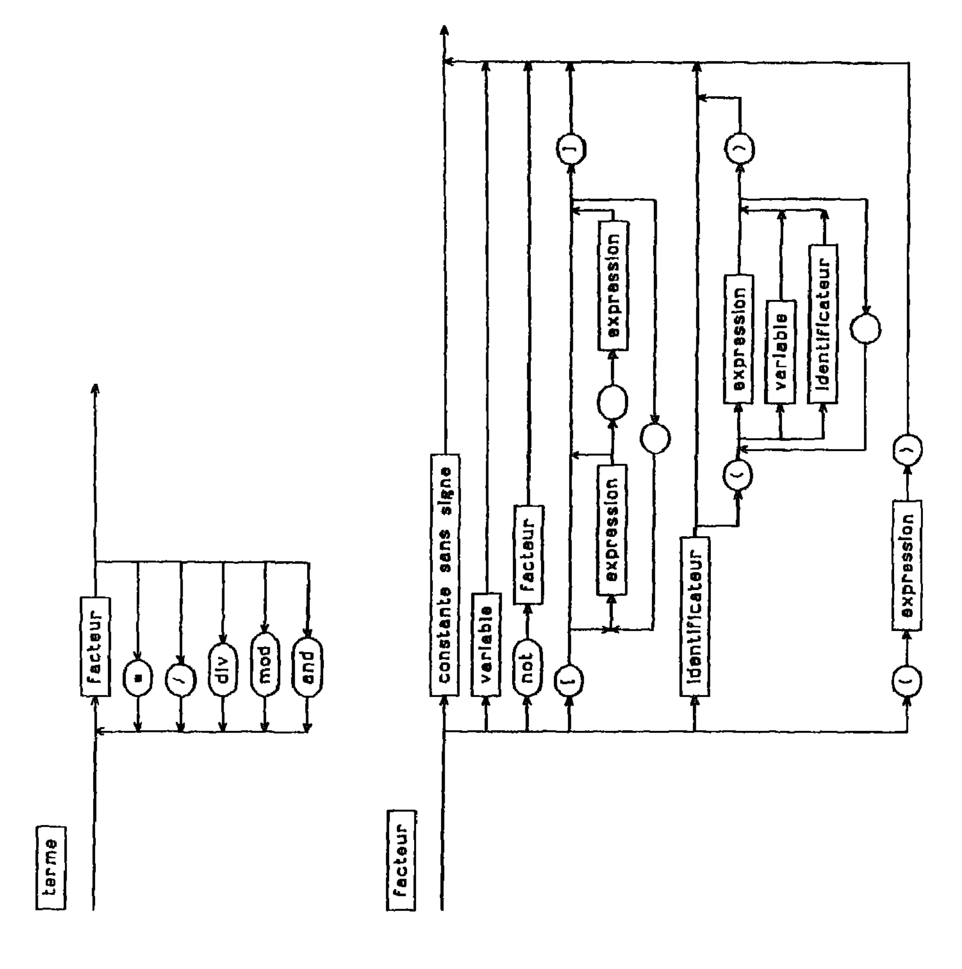

tableau ajustable
array
[
bornes
;
]
of
identificateur
tableau ajustable
packed
array
[
bornes
]
of
identificateur
bornes
identificateur
identificateur
identificateur
type
identificateur
(
identificateur
)
constante
constante
↑
identificateur
packed
array
[
type
]
of
type
record
liste de champs
end
set
of
type
file
of
type
commentaire
{
caractère
}

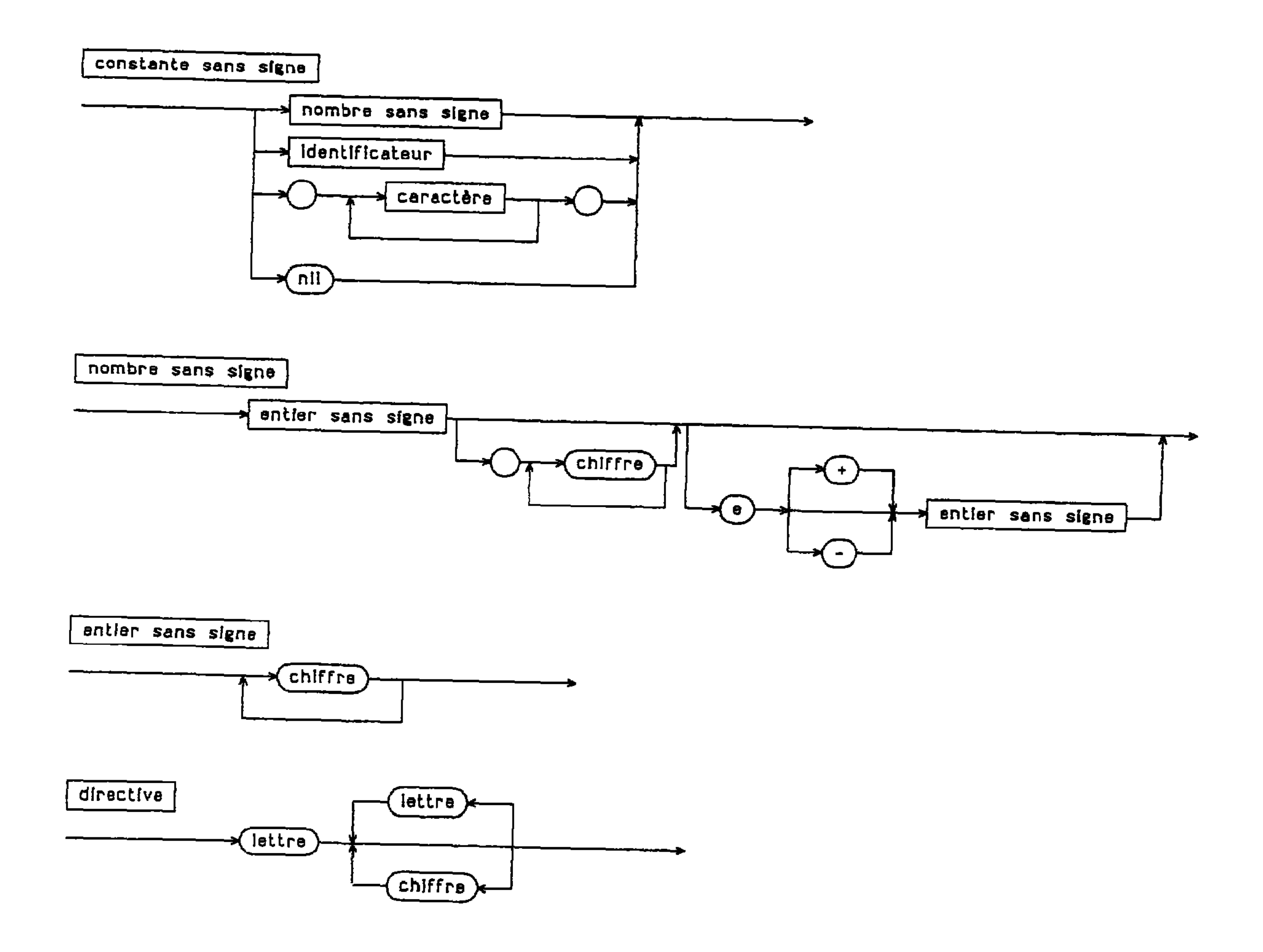

constante
+
-
nombre sans signe
identificateur
'
caractère
liste de champs
identificateur
:
type
;
case
identificateur
:
identificateur
of
constante
:
(
liste de champs
)
;
identificateur
lettre
lettre
chiffre



énoncé
étiquette
variable
identificateur
:=
expression
begin
énoncé
end
if
expression
then
énoncé
else
énoncé
case
expression
of
constante
énoncé
end
for
identificateur
:=
expression
to
downto
expression
do
énoncé
while
expression
do
énoncé
repeat
énoncé
until
expression
goto
étiquette
with
variable
do
énoncé
identificateur
(
identificateur
variable
,
)


variable
identificateur
[
expression
]
identificateur
↑
expression
expression simple
=
<>
<
<=
>
>=
in
expression simple
expression simple
+
-
terme
+
-
or


terme
facteur
*
/
div
mod
and
facteur
constante sans signe
variable
not
facteur
[
expression
expression
]
identificateur
(
expression
variable
identificateur
)
(
expression
)



constante sans signe
nombre sans signe
identificateur
caractère
nil
nombre sans signe
entier sans signe
chiffre
e
+
-
entier sans signe
entier sans signe
chiffre
directive
lettre
lettre
chiffre

ملحق 3 : تمثيل مادّي

رموز خاصة

```
+    −    ★ / = ' [ ]  .  ,
;    :    ↑ ( )  < >  <=  >=  <>
:=  espace { }
```

(espace : تباعد)

تمثيلات إضافية

```
↑  ˆ   o   @
       u
[  (.        {  (★
]  .)        }  ★)
```

كلمات دليليّـة

A	**and**		**end**	**N**	**nil**	**S**	**set**
	array	**F**	**file**		**not**	**T**	**then**
B	**begin**		**for**	**O**	**of**		**to**
C	**case**		**function**		**or**		**type**
	const	**G**	**goto**	**P**	**packed**	**U**	**until**
D	**div**	**I**	**if**		**procedure**	**V**	**var**
	do		**in**		**program**	**W**	**while**
	downto	**L**	**label**	**R**	**record**		**with**
E	**else**	**M**	**mod**		**repeat**		

لا يمكن أن يكون لدى أيّ معرِّف ، كتابة الكلمات الخاصة بإحدى هذه الـ 35 كلمة دليلية .

معرِّفين معرَّفين مسبقاً

يمكن إعادة التصريح عنهم في البرنامج ويكون لهم بذلك مدلول آخر .

ثوابت

false خطأ (بولي)

true صح (بولي)

maxint الأعداد الصحيحة محدّدة بـ maxint ... maxint —

أنواع

boolean	= (false, true)
char	لُعب سمات ، يتعلق بالحاسوب (ملحق 1)
integer	صحيح
real	حقيقي
text	سجل نص . مثلاً : input وoutput

متغيرات

input	سجل موحّـد حيث يتم قراءة المعطيات
output	سجل موحّـد حيث يتم كتابة النتائج

توجيهات

forward	يفصل عنوان وجسم الإجراء : تكرارية متقاطعة
external	يحدد بأن جسم الإجراء أو الدالة هو خارجي عن البرنامج : « تصريف منفصل» ليس بالضرورة أن يعرَّف الـ external في كل حاسوب

دوالّ

abs (x)	(x صحيح أو حقيقي ، نتيجة من نفس النوع)
artan (x)	قوس ظل x
chr (x)	سمة العدد الترتيبي x (ملحق 1)
cos (x)	جيْب التمام للزاوية x
eof (f)	بولي ، نهاية السجل
eoln(f)	بولي ، نهاية السطر في سجل نص
exp(x)	e^x
ln (x)	لوغاريتم نيبري لِـ x
odd (x)	بولي ، مفرديّة العدد الصحيح x (صحّ إذا x مفرداً)
ord (x)	العدد الترتيبي لِـ x (من نوع ترتيبي)
pred (x)	سَلَف x (من نوع ترتيبي)
round (x)	قيمة صحيحة مدوّرة للعدد الحقيقي x (العدد الصحيح الأقرب)
sin (x)	جيب الزاوية x
sqr (x)	x^2 (عدد صحيح أو حقيقي ، نتيجة من نفس النوع)
sqrt (x)	$\sqrt{x}$
succ (x)	خَلَفْ x (من نوع ترتيبي)
trunc (x)	القسم الصحيح للعدد الحقيقي x

إجراءات

dispose (x)	تحرِّر المتغير المدلَّل عليه بـ p
get (f)	قدِّم النافذة f ↑ واقرأ
new (p)	خلق متغيّـر مدلَّل عليه
pack (...)	رصّ
page (f)	تخطّي الصفحة
put (f)	اكتب ، وقدّم النافذة f ↑
read (f, v)	V : = f ↑ ; get (f)
readln (f)	ركّـز النافذة في بداية السطر التالي
reset (f)	أعِد تركيز السجل عند بدايته ، الشأن معاينة
rewrite (f)	أتلف السجل ، انتقل إلى الشأن تناتج
unpack (...)	تفكيك
write (f, c)	f ↑ : = e ; put (f)
writeln (f)	اكتب نهاية سطر

ملحق 4 : مشتقات فرنكوفون (بالفرنسية) مع ترجمتها

كلمات دليلية

A and	et	و	N nil	nil	صفر
array	tableau	جدول	not	non	لا
B Begin	début	بدء	O of	de	من
C case	cas	حالة	or	ou	أو
Const	const	ثابت	P packed	paquet	مرصوص
D div	div	قسمة صحيحة	procedure	procédure	إجراء
do	faire	إفعل	program	programme	برنامج
dounto	bas	أسْفَل	R record	article	فقرة
E else	sinon	وإلاّ	repeat	répéter	كرِّر
end	fin	نهاية	S set	ensemble	مجموعة
F file	fichier	سجل	T then	alors	إذن
for	pour	لِـ	to	haut	إلى
function	fonction	دالّة	type	type	نوع
G goto	allera	إذهب إلى	U until	jusque	حتى
I if	si	إذا	V var	var	متغير
in	dans	في	W while	tantque	طالما
L label	étiquette	وسم	whith	avec	مع
M mod	mod	شأن			

معرَّفين معرَّفين مسبقاً

ثوابت

false	faux	خطأ	true	vrai	صح
maxint	entmax	تحديد للأعداد الصحيحة			

أنواع

boolean	booléen	بولي	real	réel	حقيقي
char	car	سمة	text	texte	نص
integer	entier	صحيح			

متغيرات

input	entrée	دَخْلُ	output	sortie	خَرْجٌ

توجيهات

forward	plus loin	الأبعد	external	externe	خارجي

دوال

abs	abs	قيمة مطلقة	ord	ord	عدد ترتيبي
arctan	arctan	قوس ظل	pred	pred	سَلَف
chr	carac	سمة	round	arrondi	مُدوَّر
cos	cos	جيب التمام	sin	sin	جيب
eol	fdf	نهاية السجل	sqr	carré	مربّع
	eoln	fdln نهاية السطر	sqrt	rac 2	جذر تربيعي
exp	exp	أسِّي	succ	succ	خَلَفْ
ln	ln	لوغاريتم نبيري	trunc	tronc	قسم صحيح
odd	impair	مفرد			

إجراءات

dispose	libérer	حرِّر	read	lire	إقرأ
get	prendre	خُذْ	readln	lireln	إقرأ سطراً
new	créer	أخْلق	reset	relire	أعد القراءة
pack	tasser	كوِّم	rewrite	récrire	أعد الكتابة
page	page	صفحة	un pack	détasser	فكِّك
put	mettre	ضَع	write	écrire	اكتب
			writeln	écrireln	اكتب سطراً

ملحق 5 مَراجِعْ

يوجد تعريف للغة الباسكال في

* النظم **Langage de programmation - pascal** (1984) : AFNOR Z 65 - 300
الذي هو ترجمة للنظم ISO رقم 7185 . إنه مرجع كامل ودقيق ، لكنه موجّه إلى المنفّذ أكثر منه للمستعمِل : التعريف الأول للغة الباسكال أعْطي من قبل ن. ويرث :

* **K. Jensen, N. Wrth; Manuel de l'utilisateur Pascal;** Masson (1978)
(بالفرنسية) (ترجمة لِـ (Revised Report , User Manual)
ومتمَّم بمقال يعرض تعريفاً صورياً للغة ، فإذن صعب القراءة :

* C.A.R. Hoare, **N. Wirth; An Axiomatic Definition of the Programming Language Pascal;** Acta Informatica 2, 335- 355 (1973)
(بالإنكليزية) لغة الباسكال متأتّية من أفكار مطروحة في :

* E.W. Dijkstra, C.A.R. Hoare, O. Dahl; **Structured Programming** Academic Press (1972)
(بالإنكليزية) ومتمَّمة على شكل تطبيقي أكثر بطريقة البرمجة بالدقة المتتالية :

* N. Wirth ; **Introduction à la Programmation systématique; Masson (1976)**
(بالفرنسية) نجد وصفاً كاملاً للطريقة الإستنتاجيّة في :

* A. Durcin; Programmation : Tome 1, du problème à l'algorithme Tome 2, de l'algorithme au Programme; Dunod (1984)
(بالفرنسية) كثير من أمثلة البرمجة في الباسكال مشروحة في :

* P. G. Grogono; **Programming in Pascal; Addison Welsen (1978)**
(بالإنكليزية) . عَمَلٌ جيد وسهل حول فن البرمجة الجيدة هو :

* H.F. Ledgard; **Proverbes de Programmation;** Dunod (1978)
(بالفرنسية) . بينما يهتم الكتاب التالي أكثر بالمظهر التقني لمجمّعات المعطيات والخوارزمات :

* N. Wirth; **Algorithmes + Data Structures = Programs;** Prentice Hall (1976)
(بالإنكليزية) . هذا الأخير يتناول مسائل التصريف المُعالَجة بشكل مفصّل أكثر في :

* B. Levrat, D. Thalmann; **Conception et inplantation de langages de Programmation : une introduction à la compilation,** Gaëtan Morin
(بالفرنسية) . مثال بسيط لمصرّف الباسكال مشروح في المقالتين :

K.M. Chung, H. Yuen; **A Tiny Pascal Compiler;** BYTE 3, 9 (Sept. 78) - 13, *
10 (octo. 78)

نجـد معلومات حـول إستعمالات البـاسكـال وتـطوره في المجلة **Sigplan Notices**
(شهرية بالإنكليزية) للـ ACM ، والمجلة المخصَّصة للغة الباسكال :
(تصدر كل ثلاثة أشهر بالإنكليزية) **Pascal News**

* أخيراً على كل مبرمج قراءة :

* (صدرت ثلاثة أجزاء منه بالإنكليزية) ؛ D. Knuth; **The Art of Computer Programming;** Addison Wesley (1968).

ملحق 6 : تمدّدات

يرجع نجاح لغة الباسكال في جزء كبير منه إلى بساطتها (نسبة إلى لغات أخرى) . لكن كذلك هذه البساطة هي التي تحمل كل مستعمل على إدخال تمدداته الخاصة ، بشكلٍ فوضوي في بعض الأحيان ينتج عنه لغة لا تحمل من الباسكال أو تكاد إلّا الإسم التجاري . إلّا أن هناك إجماعاً يُستخلص حول بعض التمددات :

- إضافة جزء else أو otherwise على العبارة الحالة (cas) ، جامعةً بشكل ضمني كل الحالات غير المحدّدة بثوابت الحالة ؛
- إضافة فترات ثوابت الحالة ، في العبارة الحالة ؛
- تصريف منفصل ؛ إستقدام وتصدير ثوابت وإجراءات ؛
- نوع سجل ذي نيل مباشر (متعلق بنظام التشغيل) ؛
- تدميث متغيرات البرنامج عند التصريح ،
- متغيرات دائمة في الإجراءات ، تحفظ قيمها من تنشيط لآخر ؛
- ثوابت مركّبة (جدول ، فقرة ، . . .) ؛
- تطبيق الدالة ord على الأدلّاء (pointeurs) ؛
- نوع سلسال سمات ذي طول متغيِّر ؛
- دالة تحويل عامّة بين الأنواع ؛
- ثوابت ثمانيّة أو سادس عشريّة
- نوع ألفا (alpha) = (1... 10 أو) Packed array [1..8] of char ؛
- بلوغ تعليمات لغة الآلة .
- إلخ . . .

في نهاية تشرين أول من العام 1985 ، قامت الـ ISO (المنظمة العالمية لتوحيد القياسات) بالتصويت ، من قبل الدول الأعضاء فيها ، على مشروع دراسة النظم « تمددات الباسكال » . وذلك بهدف تجنّب إستمرار الخلافات الحاضرة ، المراد من ذلك هو التعريف السريع لمستوى جديد للغة ، متساوق مع المستويات الحالية 0 و1 وحاوٍ على عدد من التمددات :

● معايير : بهدف التعريف المنفصل لأجزاء البرنامج وعدم جمعها في برنامج كامل إلّا في لحظة التنفيذ . سيسمح هذا المفهوم بخلق مكاتب معايير ، تحوي الإجراءات والدوال التي يستطيع كل برنامج إستعمالها دون إعادة كتابتها ، ودون الحاجة إلى معرفة تفاصيلها ، بل الحاجة فقط إلى معرفة عملها الوظائفي وطريقة مناداتها . بالنسبة لُمبرمج المعيار الذي ليس بحاجة إلى معرفة تفاصيل البرنامج المُستعمِل ، فعليه من جهته فقط

معرفة الوسائط المفروض مبادلتها مع هذه البرامج والمعالجة الواجب إتمامها . مثلاً يمكن أن يحتوي معيار معين على الإجراءات والدوال المصرّحة حالياً مسبّقاً : abs, cos, sin . . .

● **سلاسل سمات** : نوع جديد ، مؤلف من سلاسل سمات بحيث يمكن لحجمها التغيُّر خلال تنفيذ البرنامج الذي سيُطبَّق عليه العمليات التالية :

- تنضيد (مِن يُنَضِّد) (concaténation) ، منوّطة + ، لِلَّصقْ طرفاً بطرف ، لسِلسالين .
- تقسيم (indexation) : S [i] هو السمة i (ieme) من السلسال S
- دالّة length (s) تَحسِب الطول المتداول للسلسال S
- دالة Capacity (s) تحسب الطول الأقصى لسلسالٍ مُتساوق بالنسبة للتعيين مع السلسال s
- دالة Position (s1, s2, i) ، وتنتج القيمة 0 إذا لم يظهر السلسال s1 في السلسال s2 ، أو تعطي موقع أوّل ظهور للسلسال s1 في السلسال s2 إنطلاقاً من s2 [i]
- إجراء insert (s1, s2, i) ، يولج السلسال s1 في السلسال s2 إنطلاقاً من s2 [i]
- إجراء delete (s, i, n) يحذف n سمة في السلسال s إنطلاقاً من s [i]
- إجراء extract (s1, s2, i, n) يكوِّن السلسال s2 من n سمة مأخوذة في السلسال s1 إنطلاقاً من s2 [i]

● **سجلات ذوات نَيْل مباشر** : النوع الجديد

file [n] of..

ستتوافق مع سجل من n مركّب ، بحيث يمكن بلوغ كل مركّب مباشرة (بواسطة get أو put) .

● **سجلات خارجية** : حالياً وحدها وسائط البرنامج تسمح بإقامة علاقة بين سجلات البرنامج والسجلات المعروفة لنظام التشغيل . المقصود هو إمكانية إقامة هذه العلاقة خلال تنفيذ البرنامج ، عن طريق إعطاء الإسم نظام السجل كوسيط لإجراء الفَتح reset أو rewrite .

● **مخططات** : ستكون أنواع جداول جديدة بحيث يتم تعريف فترة الدليل خلال التنفيذ ، هذا الحلّ القاضي بتوسيط (مِن وسيط) الأنواع هو أكثر شمولية من الجداول الضبيطة .

هذه التمددات ، في حال إتمام تنظيمها ، تحمل إمكانيات جديدة إلى لغة الباسكال ، مع المحافظة على صحّة البرامج الموجودة سابقاً : إنها لا تعرِّف لغة جديدة .

غير أنّ عدة لغات تحمل من قبل تمددات إلى الباسكال ، لكن دون التأكد من

التواصُل مع اللغة الحالية . هكذا فإن البرمجة في لغة الـ Ada أو الـ Modula 2 تسمح بتناول حقولاً سيكون الباسكال فيها غير فعّال ، مع المحافظة على نحوٍ قريب من الباسكال .

فهرست